



**TRAVELING SALESMAN PROBLEM
FOR SURVEILLANCE MISSION
USING PARTICLE SWARM OPTIMIZATION**

THESIS

Barry R. Secrest, Captain, USAF

AFIT/GCE/ENG/01M-03

DEPARTMENT OF THE AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

20010706 129

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the U.S. Air Force, Department of Defense, or U. S. Government.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE (DD-MM-YYYY) 20-03-2001		2. REPORT TYPE Master's Thesis		3. DATES COVERED September 2000 - March 2001	
4. TITLE AND SUBTITLE Traveling Salesman Problem For Surveillance Mission Using Particle Swarm Optimization				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
				5d. PROJECT NUMBER	
6. AUTHOR(S) Secrest, Barry R.				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Building 640 WPAFB OH 45433-7765				8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCE/ENG/01M-03	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Raj P. Malhotra AFRL/SNAT, Bldg 106 2241 Avionics CI Building 620 Rm WPAFB OH 45433 785-1115 x 4291				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AILABILITY STATEMENT APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED					
13. SUPPLEMENTARY NOTES Dr. Gary B. Lamont, Professor, (937)255-3636 x4718, Gary.Lamont@afit.edu					
14. ABSTRACT The surveillance mission requires aircraft to fly from a starting point through defended terrain to targets and return to a safe destination (usually the starting point). The process of selecting such a flight path is known as the Mission Route Planning (MRP) Problem and is a three-dimensional, multi-criteria (fuel expenditure, time required, risk taken, priority targeting, goals met, etc.) path search. Planning aircraft routes involves an elaborate search through numerous possibilities, which can severely task the resources of the system being used to compute the routes. Operational systems can take up to a day to arrive at a solution due to the combinatoric nature of the problem. This delay is not acceptable because timeliness of obtaining surveillance information is critical in many surveillance missions. Also, the information that the software uses to solve the MRP may become invalid during computation. An effective and efficient way of solving the MRP with multiple aircraft and multiple targets is desired. One approach to finding solutions is to simplify and view the problem as a two-dimensional, minimum path problem. This approach also minimizes fuel expenditure, time required, and even risk taken. The simplified problem is then the Traveling Salesman Problem (TSP). While the TSP directly relates to the surveillance mission, it also has other applications. It is the most notorious NP-Complete					
15. SUBJECT TERMS Genetic Algorithm, Evolutionary Computation, Traveling Salesman Problem, Particle Swarm Optimization					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Dr. Gary B. Lamont
U	U	U	UU	131	19b. TELEPHONE NUMBER (Include area code) (937)255-3636 x4718

AFIT/GCE/ENG/01M-03

**TRAVELING SALESMAN PROBLEM
FOR
SURVEILLANCE MISSION
USING
PARTICLE SWARM OPTIMIZATION
THESIS**

Presented to the Faculty of the School of Engineering and Management
Of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of Master of Science in Electrical Engineering

Barry Secrest, B.S.

Capt, USAF

20 March 2001

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AFIT/GCE/ENG/01M-03

TRAVELING SALESMAN PROBLEM
FOR
SURVEILLANCE MISSION
USING
PARTICLE SWARM OPTIMIZATION
THESIS

Barry Secrest, Capt, USAF

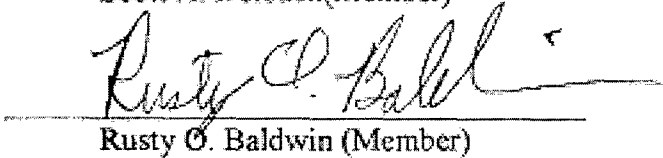
Approved:


Gary B. Lamont (Chairman)

1 MAR 01
date


Scott A. Deloach (Member)

6 Mar 01
date


Rusty C. Baldwin (Member)

6 Mar 01
date

Acknowledgements

I would like to thank the members of my thesis committee, Dr. Gary B. Lamont, Dr. Maj. Scott A. Deloach, and Maj. Rusty O Baldwin for their guidance and assistance. . . .

Barry Secrest

Table of Contents

<u>ACKNOWLEDGEMENTS</u>	V
<u>TABLE OF CONTENTS</u>	VI
<u>LIST OF TABLES</u>	IX
<u>ABSTRACT</u>	X
1 INTRODUCTION	1
1.1 BACKGROUND	1
1.2 APPROACH.....	2
1.3 THESIS OVERVIEW.....	4
2 LITERATURE REVIEW	6
2.1 UNMANNED COMBAT AIR VEHICLE (UCAV) RESEARCH.....	6
2.1.1 Predator.....	6
2.1.2 Global Hawk.....	6
2.1.3 Dark Star.....	8
2.1.4 Air Balloons.....	8
2.1.5 Miniature UCAVs.....	9
2.1.6 Use and Design Research	9
2.1.7 Unique Engineering Design Issues	9
2.2 THE TRAVELING SALESMAN PROBLEM (TSP) FOR UCAV	10
2.2.1 Formal TSP Description	10
2.2.2 Formal Problem Requirements Specification Form.....	11
2.2.3 Formal Algorithm Specification Form for PSO solving TSP	12
2.2.4 Mapping the Algorithm to the Problem	13
3 PROPOSED ALGORITHM FOR FINDING SOLUTIONS TO THE TSP	14
3.1 PSO_AS IMPLEMENTATION	14
3.2 MOVING THE SWARM	14
3.2.1 Permutations	14
3.2.2 Tour Building.....	15
3.2.3 Random Choice of Method.....	20
3.2.4 The Default Selection Method in the PSO_AS Operator.....	20
3.3 INITIALIZATION METHODS	21
3.4 HILL-CLIMBING AFTER MOVING	21
3.5 RE-HOPE METHODS.....	21
3.6 SUMMARY	22
4 DESIGN OF EXPERIMENTS	23
4.1 GENERAL TESTING CONSIDERATIONS	23
4.1.1 TSP Standard Test Suite	23
4.1.2 Complexity and Classification of TSP Problem	24
4.1.3 Collection of Data.....	24
4.1.4 Data Reduction/Presentation.....	25
4.2 INDIVIDUAL TEST SPECIFICS.....	25
4.2.1 Baseline Data	26
4.2.2 PSO_AS Parameter Tuning Tests.....	27

4.3	SUMMARY OF TESTS.....	29
5	RESULTS AND ANALYSIS	31
5.1	EXPERIMENT1 - LK BASELINE, EXPERIMENT 2 - PSO_AS BASELINE	31
5.2	EXPERIMENT 3 – AS BASELINE.....	33
5.3	EXPERIMENT 4 – INVER-OVER BASELINE.....	33
5.4	EXPERIMENT 5 – COURSE TUNING PSO_AS: 10% DEFAULT “TRUST”, EXPERIMENT 6 – COURSE TUNING PSO_AS: 10% LOCAL “TRUST”	34
5.5	EXPERIMENT 7 – GREEDY PSO_AS: 10% DEFAULT “TRUST”	36
5.6	EXPERIMENT 8 – GREEDY PSO_AS: 10% LOCAL “TRUST”	37
5.7	EXPERIMENT 9, 10, 11, AND 12 – FINE TUNING PSO_AS: 85,90,95% GLOBAL “TRUST” WITH OR WITHOUT GREEDY METHOD	38
5.8	EXPERIMENT 12 – PSO_AS TUNED FOR QUALITY OF SOLUTION.....	40
5.9	EXPERIMENT 13 – PSO_AS TUNED FOR SPEED	41
5.9.1	Kruskal-Wallis Test	43
5.10	EXPERIMENT 14 – PARALLELIZATION OF PSO_TSP	44
5.11	EXPERIMENT 15 – PSO_TSP vs PSO_AS	44
5.12	EXPERIMENT 16 –TUNED PSO_AS vs PUBLISHED RESULTS OF INVER-OVER	44
5.13	EXPERIMENT 17 –TUNED PSO_AS vs AFIT RESEARCH.....	44
5.14	SUMMARY OF RESULTS AND ANALYSIS.....	45
6	CONCLUSIONS AND RECOMMENDATIONS.....	46
6.1	PERFORMANCE OF PSO_AS	46
6.2	ALTERNATIVES TO PSO_AS.....	47
6.3	TUNING OF PSO_AS	47
6.4	IMPROVEMENTS TO PSO_AS.....	48
7	REFERENCES	49
7.1	TSP	49
7.2	UAV	49
7.2.1	Predator.....	49
7.2.2	Global Hawk.....	49
7.2.3	Dark Star.....	50
7.2.4	Air Balloons.....	50
7.2.5	Miniature UCAVs.....	50
7.2.6	Miscellaneous	50
7.2.7	Other Research.....	50
7.3	SWARM.....	51
7.3.1	Representation/Explanation	51
7.3.2	Application	51
7.3.3	Tuning.....	51
7.3.4	Ant System.....	51
7.4	EVOLUTIONARY COMPUTATION	51
A	CURRENT HEURISTIC AND STOCHASTIC APPROACHES TO SOLUTION	53
B	EVOLUTIONARY COMPUTATION.....	62
C	TESTING SCRIPTS.....	76
D	EXCEL SPREADSHEETS OF RAW DATA AND CHARTS	79
E	HETEROGENEOUS IMPLEMENTATION OF A PARTICLE SWARM OPTIMIZATION... 81	
F	COMMUNICATION IN PARTICLE SWARM OPTIMIZATION	82

List of Figures

Figure 1 - Predator	6
Figure 2 - Global Hawk on Ground	7
Figure 3 - Global Hawk Flying	7
Figure 4 - Dark Star	8
Figure 5 - Air Balloon	8
Figure 6 - X_t	17
Figure 7 - $P_{vg,t}$	17
Figure 8 - $P_{ig,t}$	17
Figure 9 - Step 1	18
Figure 10 - Step 2	18
Figure 11 - Step 3	18
Figure 12 - Step 4	18
Figure 13 - Step 5	19
Figure 14 - Step 6	19
Figure 15 - Step 7; X_{t+1}	31
Figure 16 - PSO_AS and LK Baseline Solutions	32
Figure 17 - PSO_AS and LK Baseline Average Time (sec)	32
Figure 18 - PSO_AS and LK Baseline Average Time (sec): Cities < 800	32
Figure 19 - Quality of Solution vs Course Global Percentage (Default = 10%)	34
Figure 20 - Quality of Solution vs Course Global Percentage (Neighborhood = 10%)	34
Figure 21 - Average Percentage from Optimal and Average Time (sec) vs Course Global Percentage (Default = 10%)	35
Figure 22 - Time/Generation (sec) vs Course Global Percentage (Default = 10%)	35
Figure 23 - Time/Generation (sec) vs Course Global Percentage (Neighborhood = 10%)	36
Figure 24 - Greedy Quality of Solution (Default = 10%)	36
Figure 25 - Greedy Time (sec) vs Global Percentage	37
Figure 26 - Quality of Solution: Fine Tuning Global Trust	38
Figure 27 - Time: Fine Tuning Global Trust	39
Figure 28 - Time(sec)/Generation: Fine Tuning Global Trust	40
Figure 29 - LK Vs PSO_AS Quality Tuned: Quality of Solution Comparison	40
Figure 30 - LK Vs PSO_AS Quality Tuned: Time(sec)	41
Figure 31 - PSO_AS Quality Vs PSO_AS Speed: Quality of Solution Comparison	42
Figure 32 - PSO_AS Speed Vs PSO_AS Quality Tuned: Time(sec)	42
Figure 33 - Pseudocode for Evolutionary Algorithm	64

List of Tables

Table 1 - Formalized Table of the Problem Domain.....	12
Table 2 - Formalized Table of the PSO for TSP Algorithm.....	12
Table 3- Comparison of PSO-AS with Published Results of Inver-Over	44
Table 4 - PSO_AS vs Published Results of GTTS [Hall].....	44
Table 5 - Biological and Evolutionary Algorithm Terminology.....	62
Table 6 - A classification of various Evolutionary Algorithms.....	73

List of Equations

Equation 1 - Kruskal-Wallis Test	43
Equation 2 - Calculating a Single Particle's New Velocity.....	59
Equation 3 - "Moving" a Single Particle in a Swarm.....	59
Equation 4 - Evolutionary Algorithm	65

Abstract

The surveillance mission requires aircraft to fly from a starting point through defended terrain to targets and return to a safe destination (usually the starting point). The process of selecting such a flight path is known as the Mission Route Planning (MRP) Problem and is a three-dimensional, multi-criteria (fuel expenditure, time required, risk taken, priority targeting, goals met, etc.) path search. Planning aircraft routes involves an elaborate search through numerous possibilities, which can severely task the resources of the system being used to compute the routes. Operational systems can take up to a day to arrive at a solution due to the combinatoric nature of the problem. This delay is not acceptable because timeliness of obtaining surveillance information is critical in many surveillance missions. Also, the information that the software uses to solve the MRP may become invalid during computation. An effective and efficient way of solving the MRP with multiple aircraft and multiple targets is desired. One approach to finding solutions is to simplify and view the problem as a two-dimensional, minimum path problem. This approach also minimizes fuel expenditure, time required, and even risk taken. The simplified problem is then the Traveling Salesman Problem (TSP).

While the TSP directly relates to the surveillance mission, it also has other applications. It is the most notorious NP-Complete optimization problem, therefore advances in finding solutions to it are applicable to all other NP-Complete problems.

Both Particle Swarm Optimization (PSO_TSP) and the Ant System (AS) have been shown to provide good solutions to the TSP. This thesis presents a new algorithm (PSO_AS), a synthesis of PSO and AS. Efficient implementation in C++ and extensive testing was done to demonstrate the algorithm's effectiveness.

TRAVELING SALESMAN PROBLEM
FOR
SURVEILLANCE MISSION
USING
PARTICLE SWARM OPTIMIZATION

1 Introduction

1.1 Background

Global Hawk is an Unmanned Combat Air Vehicle (UCAV). Since it is unmanned, it can be built smaller and thus have a smaller radar signature. Its mission is reconnaissance / surveillance. It flies at a near-constant height above enemy territory and gathers data using various sensors (see 7.2.2).

Mission planning for Global Hawk is computationally challenging. In what way do you fly the Global Hawk over the territory to obtain the data? It is desirable to travel the shortest path, since that path allows us to obtain the data swiftly, minimize flight time thus minimizing the possibility of being shot down by enemy fire, as well as consuming less fuel. Given a set of up to 2000 coordinates (either from earlier satellite or Global Hawk mission reconnaissance) of the desired targets, mission planning is then easily mapped to the ever-popular Traveling Salesman Problem (TSP) – a well-known NP-Complete problem. Unfortunately, complexity increases with the addition of real-world requirements and constraints. For example, certain targets must be flown over within a specific time window, or may be optional to visit (with an associated priority/cost function). Enemy radar, or even mountainous terrain may make certain paths undesirable. There may be several Global Hawks available to cover the terrain. Which

targets should be assigned to which Global Hawk? Finally, to be able to plan and fly a three-dimensional path of least cost, current weather conditions and flying specifications (fuel level, fuel consumption, max speed, turning radius for various speeds) must all be considered. The Mission Routing Problem (MRP) is defined by the addition of these factors. See [Harder] for an excellent background review on MRP.

A tiered approach to finding solutions to this multi-objective, multi-layered NP-Complete problem has been shown to produce good results [Harder]. In such an approach, many good solutions to a simpler problem (TSP) are further analyzed against the multi-objective problem (MRP). Thus, the first step in finding a good solution to MRP is to find good solutions to TSP.

Both Particle Swarm Optimization (PSO_TSP) [Clerc] and the Ant System (AS) [Dorigo] have been shown to produce good solutions to the TSP. They are modeled after the way swarms of insects, though individually not very capable, can produce nearly optimal solutions to highly complex problems. Experiments have shown that ants will follow a nearly optimal path to food [Beckers]. A new combination algorithm (PSO_AS) inspired by both PSO_TSP and AS is developed.

1.2 Approach

A straightforward, scientific approach is applied. The problem domain is analyzed (see 2.2) along with current attempts at finding solutions (see Appendix A). The algorithm design is presented at a high level (see 2.2.4) and low level (detailed) (see A.10.2). The algorithm implementation is then presented [see Chapter 3]. Experimental tests and analysis are conducted for the purpose of improving performance and comparing to current attempts at solution [see Chapter 4 and 5].

Specifically, this research investigation consists of the following tasks:

- Analyze the problem domain
- Alter the PSO_TSP code to utilize the MPICH implementation of the Message Passing Interface (MPI) as a means of communication between the processors. Implement it on a parallel computational platform to assess this approach
- Alter PSO_TSP to accept and be able to use a wide variety of [TSPLIB] formatted problems. Internally, the code was written to use a full matrix representation, but [TSPLIB] problems come in full matrix, upper diagonal matrix, lower diagonal matrix, and Euclidean format. To implement the Euclidean format, change the coordinate system from a geocentric format expressed in degree, minute, second notation with Latitude and Longitude values to a matrix of associated path cost between coordinates.
- Test and analyze the parallel PSO_TSP code
- Implement a new algorithm (PSO_AS) based on both PSO_TSP and AS. PSO_AS has the following features (see Chapter 3 for greater detail):
 - Variable (settable) parameters:
 - Size of swarm, neighborhood
 - Amount of global trust, local trust, and default trust
 - Number of iterations before “re-hope”
 - Various Swarm initialization, moving, hill-climbing, and re-hope methods
 - Greedy
 - Re-Insert
 - Opt2-3

- Lin-Kernighan
- Inver-Over
- Ant System
- Random
- Either array or linked-list data structures
- Tune the algorithm with respect to all of the features
- Test PSO_AS to obtain performance metrics (speed and quality of solution) and compare it against currently used methods to determine efficiency and effectiveness.

This research effort includes design of experiments, results of the experiments, and qualitative and quantitative analysis. Conclusions and recommendations based on the qualitative and quantitative data analysis are presented. Additionally, insights for code improvements and areas for continued research are discussed.

1.3 Thesis Overview

This chapter provides an introduction to the surveillance mission, TSP domain, the scope of the thesis investigation, and the approach taken for solving the problem. The remainder of the thesis consists of seven chapters and six appendices. Chapter II is a literature review of UCAV research as well as the TSP problem domain and high-level algorithm domain. Chapter III details the PSO_AS algorithm low-level design and implementation issues. Chapter IV describes the design of experiments. Chapter V is testing, results, and analysis. Chapter VI includes conclusion remarks and recommendations for future effort.

Appendix A gives state-of-the-art attempts at finding solutions (heuristic and stochastic) including a background review of PSO_TSP (brief), AS, and Lin-Kernighan. Appendix B gives an overview of Evolutionary Computation and discusses high-level design considerations of PSO_AS in the context of Evolutionary Computation. Appendix C gives testing scripts used. Appendix D gives raw data and Excel spreadsheets obtained from tests. Appendix E and F are papers written concerning PSO_TSP and share some insight on why the new PSO_AS algorithm was needed.

2 Literature Review

2.1 Unmanned Combat Air Vehicle (UCAV) Research

In order to logically present the material, we focus on some currently available hardware (starting with Air Force designed and proceeding to commercially available) along with its specifications and the mission for which it was designed. We then proceed with a review of some research articles and a discussion of some of the unique engineering design issues that UCAVs must address.

2.1.1 *Predator*

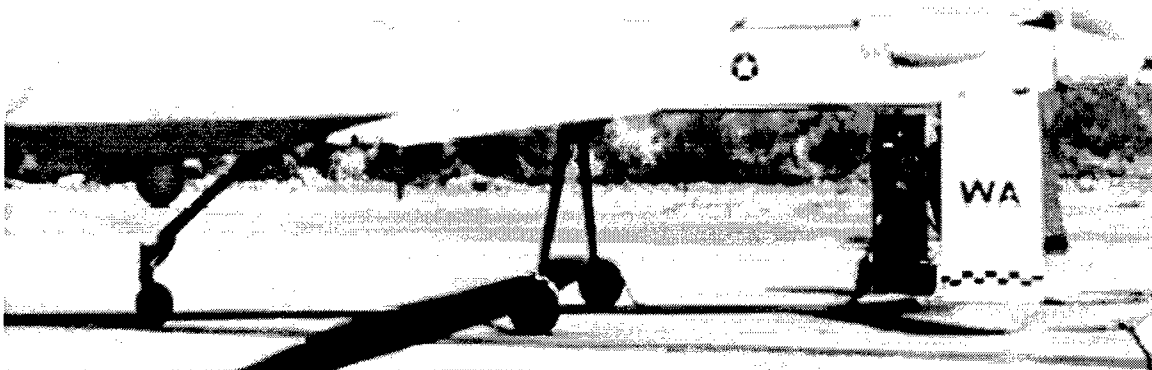


Figure 1 - Predator

The RQ-1 Predator was designed for the medium altitude reconnaissance mission. It is capable of flying 24 to 30 hours. It flies at an upper altitude of 25,000 feet, but usually flies at the 10-15,000 feet level. It has a cruise speed of 84 mph (70 knots) with an upper limit of 140 mph (120 knots). It is a fielded product with six systems at a cost of 25 million each in the inventory [see 7.2.1].

2.1.2 *Global Hawk*

Global Hawk was designed for the high-altitude, long endurance reconnaissance mission. It is capable of flying a maximum of 40 hours. It flies at an upper altitude of 65,000 feet, and speeds of up to 340 knots. It has flown its first mission in Bosnia, although not a fielded system. It is scheduled to be fielded in 2001 [see 7.2.2].



Figure 2 - Global Hawk on Ground

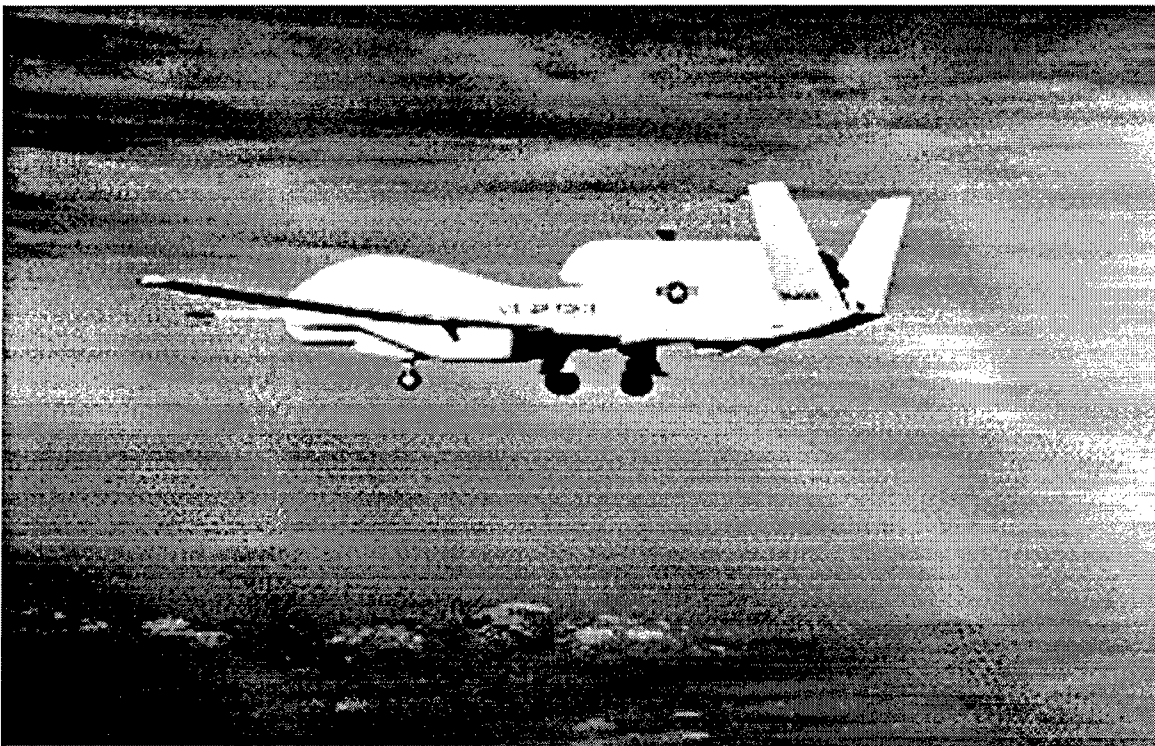


Figure 3 - Global Hawk Flying

2.1.3 *Dark Star*

Dark Star is a stealthy platform designed to perform reconnaissance in a high-threat environment. It is capable of flying 12 hours. It flies at an altitude above 45,000 feet at speeds near 250 knots. The project was cancelled for unspecified reasons [see 7.2.3].



Figure 4 - Dark Star

2.1.4 *Air Balloons*

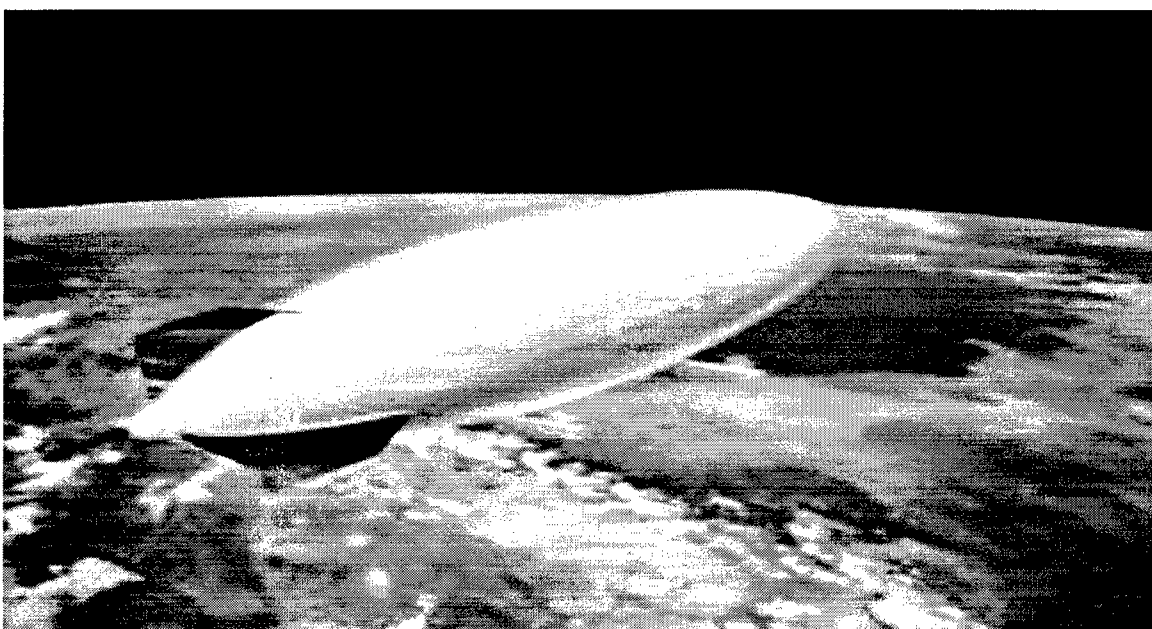


Figure 5 - Air Balloon

While Air Balloons may be in the public eye as nothing more than something appearing at sports events and providing fancy advertising, they have great potential. Commercial UCAV Air Balloons have been deployed for gathering weather information, reconnaissance, and when placed in a geo-synchronous flight path, to replace very costly communication satellite networks [see 7.2.4].

2.1.5 Miniature UCAVs

The Black Widow measures only six inches across and weighs a mere two ounces. It has a camera for real-time video downlink and flies at speed of up to 43 mph for up to 22 minutes. In a military setting, it could perform specialized reconnaissance. It could potentially be flown to keep track of a military target of interest [see 7.2.5].

2.1.6 Use and Design Research

Since UCAVs are air vehicles, they share many of the same issues as manned air vehicles. Research associated with UCAVs include scheduling, routing, using Tabu Search to solve the TSP [Harder, Kinney], and using A* in a parallel implementation to perform route planning [Sezer]. While these works are associated with UCAVs, they share with this work the characteristic of not being unique to UCAVs. Thus, design research such as sensor improvement or target recognition may be sponsored and targeted for use in UCAVs, but are certainly not unique [see 7.2.7].

2.1.7 Unique Engineering Design Issues

Unique software design issues revolve mostly around the coordinated planning, re-planning, robotic (swarm) control, and autonomous action of each UCAV. There are no direct UCAV publications in the literature on these subjects (individual companies do have internal publications), a search of “Agent” and any of the other keywords above will yield many references.

2.2 The Traveling Salesman Problem (TSP) for UCAV

The goal of this investigation is to design, implement, test and analyze an effective and efficient TSP system that produces solutions in a timely manner using PSO. Given a list of Earthly coordinates (these are the targets that a UCAV must fly over), minimize the path taken.

2.2.1 Formal TSP Description

The TSP is defined as follows: Given an $n \times n$ distance matrix $C = (c_{ij})$, find a permutation (π) , that is a member of all possible solutions $(\pi \in S_n)$ that minimizes the sum $\sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}$ [Lawler]. The Salesman must visit all cities from 1 to n exactly once in such a way to minimize the distance traveled. It builds upon the Hamilton Circuit Problem (which seeks to find *any* path where every city is visited exactly once) by the additional constraint of finding the minimum path. It has direct, real-world application to routing problems in general (mail or other kinds of delivery), and as a NP-complete problem [see Garey and Johnson] has indirect relevance to many other real-world problems. It is undoubtedly the most famous NP-complete problem.

The problem is often classified by characteristics of the distance matrix. In symmetrical TSPs, $c_{ij} = c_{ji}$, thus all values in the matrix are symmetrical about the diagonal. If it is not symmetric, it is asymmetric. In triangular TSPs (Δ TSP) $\forall i, j, k \in C : c_{ik} \leq c_{ij} + c_{jk}$, thus given the distances between any three cities, a physical triangle can be formed to represent those three cities' distances. In Euclidean TSPs, $\forall i, j \in C : c_{ij} = c_{ji} = \sqrt{(a_i - a_j)^2 + (b_i - b_j)^2}$ (i.e., the formula for distance between two points), thus the cities can be mapped to a physical (Euclidean) grid. Even though Euclidean TSPs are a small portion of all possible TSPs, it is perhaps the most relevant classification since most real-world problems mapped to the TSP are Euclidean. In fact, the TSP for the surveillance mission is Euclidean. Euclidean TSPs are both symmetric and triangular as seen from the definitions, but these characteristics alone do not imply Euclidean. By taking four Euclidean points and increasing or decreasing one (and only one) of the costs slightly so that it remains triangular and symmetric, you then have a

simple example that cannot be mapped to a Euclidean grid since all distances no longer meet the requirement. It is also possible for a matrix to be triangular and asymmetric. Other matrix classifications include Kalmanson, Demidenko, or Supnic conditions, which are trivially solved by looking at the plot of the points [see Burkard , Deineko, Demidenko].

Problem complexity is often related to matrix classification. For Kalmanson, Demidenko, and Supnic matrices, optimal solutions can be generated in polynomial time because these are “trivial” cases (i.e., for Supnic matrices, all cities are linear). Δ TSPs are easier to find good solutions for the same dimensionality because the triangular relationship is used heuristically to produce solutions. Symmetric TSPs are easier than asymmetric because a multi-city path can be reversed (inverted) without having to recalculate the path cost. Furthermore, of all possible tours, there are twice as many solutions (the forward tour and the backward tour), thus only half of the search space needs to be searched to find the optimal solution. Since they are both symmetric and triangular, Euclidean TSPs are considerably easier to find good solutions for, yet no polynomial algorithm has been discovered to find the optimal solution. In fact, Euclidean TSP alone is NP-Complete [see Young].

The Lin-Kernighan algorithm is currently the most efficient heuristic method for finding good solutions to the TSP known, but doesn't work as well with asymmetric TSPs for the reasons discussed previously. The requirement to calculate the cost of a reverse path (instead of assuming an equal cost) raises the dimensionality of the algorithm by two (i.e., finding the path-cost of all pairs in the current tour is a $O(n^2)$ problem). Perhaps the most efficient stochastic method is the inver-over operator [Tao] (see A.9), which may produce better solutions than Lin-Kernighan, but requires more time. It is likewise limited to working well only for symmetric, Δ TSP. For further information on TSP, see [Lawler].

2.2.2 Formal Problem Requirements Specification Form

Although similar to the TSP description, this table lists the real-world problem requirements. The major difference between them is that the Domain of the problem is a list of Earthly coordinates instead of a cost matrix.

Solution Space Size:	$O(N!)$ where N is the number of coordinates
Domains:	Di: (C) – a list of coordinates (x,y) Do: (C') – an ordered permutation of C such that the total distance of the Hamiltonian cycle is minimized
Objective Function:	$\text{Min } c = \sum d_{ij}$ where d_{ij} is the physical distance between adjacent elements of C'
Candidates:	C' is an ordered permutation of C
Selection Function:	Always True. Normally this would be concerned with if C' is in fact a Hamiltonian cycle, but since UCAVs are a flying vehicle we thus have a totally connected graph, and all ordered permutations of C are Hamiltonian cycles.
Feasibility Function:	Always True. Normally this would be concerned with if a Hamiltonian cycle exists, but since UCAVs are a flying vehicle we thus have a totally connected graph, and all ordered permutations of C are Hamiltonian cycles.
Solution Function:	C' is an ordered permutation of C and $\text{Min } c = \sum d_{ij}$ where d_{ij} is the physical distance between adjacent elements of C'

Table 1 - Formalized Table of the Problem Domain

2.2.3 Formal Algorithm Specification Form for PSO solving TSP

This table describes the specifications for any PSO algorithm (including PSO_TSP and PSO_AS discussed in this work) used to solve TSP.

Solution Space Size:	$O(N!)$ where N is the number of coordinates
Search Space Size:	$O(I*N)$ where I is the number of iterations. Since PSO is stochastic in nature, it searches a portion of the search space. The portion that it searches is determined by a parameter set by the user of the algorithm. The quality of the result depends on the number of iterations used.
Domains:	Di: (D_{ij}) – A square matrix representing the distances between cities i and j . Do: (C') – an ordered permutation of the indices of D such that the total distance of the Hamiltonian cycle is minimized
Objective Function:	$\text{Min } c = \sum d_{ij}$ where d_{ij} is the physical distance between adjacent elements of C'
Candidates:	C' is an ordered permutation of the indices of D . Candidates are referred to as “positions”.
Selection Function:	$I(C', V)$: where C' is a position or solution and V is a “velocity” or list of permutation steps $O(C', V, CB', VB)$: where CB' is the best position found so far and VB is a list of permutation steps to transform C' to CB'
Feasibility Function:	Always True. Normally this would be concerned with if a Hamiltonian cycle exists, but since we are dealing with a flying vehicle we thus have a totally connected graph, and all ordered permutations of the indices of D are Hamiltonian cycles.
Solution Function:	C' is an ordered permutation of C and $\text{Min } c = \sum d_{ij}$ where d_{ij} is the physical distance between adjacent elements of C'

Table 2 - Formalized Table of the PSO for TSP Algorithm

2.2.4 *Mapping the Algorithm to the Problem*

There is not a direct data structure mapping from the algorithm domain to the problem domain. In order to produce a mapping of the input of the problem to the input of the algorithm, the distance between coordinates is calculated and a symmetrical matrix representing d_{ij} is produced and used in the algorithm. This mapping of inputs is $O(n^2)$ – the number of operations needed to fill the square matrix. For output, there is a direct mapping $O(1)$ from the “city numbers” in the sequence of the tour to the original coordinates using the city number as an index to determine the coordinates.

3 Proposed Algorithm for Finding Solutions to the TSP

This chapter presents the PSO_AS algorithm. From a high level, it is the same as PSO_TSP (see Appendix A.10.2). The Swarm is initialized, then moves and hopefully improves with each iteration. If there is no improvement after several iterations, a “no-hope” or “re-hope” portion of code is executed and the Swarm is moved for more iterations. When there is no improvement and all “re-hope” methods are exhausted, the algorithm ends.

3.1 PSO_AS Implementation

PSO_AS is an object oriented C++ program. The objects in the code are Swarm, Particle, Velocity, and Position. It is a layered, hierarchical architecture. The Swarm contains an array of Particles. Each Particle contains both a Position and a Velocity. Each Position contains an array (or linked list) of both the current position and the best position found. To facilitate testing, all input variable parameters are read from a text file. All of the variations discussed in this section are configurable by the input text file.

3.2 Moving the Swarm

Because of the hierarchical nature, most of the “work” is done in objects lower than the Swarm. To move the Swarm, simply move each Particle. To move the Particle, just successively select each city and move them into the Position. Perhaps the largest and most complex section of code is the decision making process. This is the heart and soul of the algorithm, and there are many different ways to move the Swarm.

3.2.1 *Permutations*

The many “Move” variations can be subdivided into permutations and “tour building”. Permutations take a completed tour (usually the current Position) and re-order it in some way. Often, permutations are hill-climbing in nature, making minor changes and improvements in incremental steps. The permutation operators are City-Swapping, Re-Insert, 2-Opt, 3-Opt, Lin-Kernighan, Inver-Over, and Swap. While they function similar to the algorithms for which they are named, as part of moving the swarm they only produce a partial result. Whereas these referenced algorithms contain an outer loop (i.e., for all cities), these operators only perform the inner loop (i.e. for a given city).

3.2.1.1 City-Swapping

Perform a single iteration of city-swapping (see A.4) and swap two and only two cities that maximize improvement.

3.2.1.2 Re-Insert

Randomly select a city. Remove it and re-insert it (see A.2) for maximum improvement.

3.2.1.3 2-Opt and 3-Opt

Perform the cuts (2 or 3) and associated inversion/re-joining (see A.5 and A.6) for maximum improvement. Only a single step-improvement is performed.

3.2.1.4 Inver-Over

Perform a single inversion of a segment. The starting city is chosen randomly and the adjoining city is chosen via the Inver-Over algorithm (see A.9). This operator “mates” the local best Position and the local best position of a randomly selected Position in the Swarm. When the neighborhood size is zero, the local best position is the personal best and the operator performs as described in A.9. A configurable variation of this operator is to mate the local best and the global best (instead of a randomly selected) Position.

3.2.1.5 Lin-Kernighan

Perform a single variable r-Opt cut as per A.7. Two or more cuts are performed and the associated segments are re-joined in such a way for maximal improvement.

3.2.1.6 Swap

Two cities in the given Position are randomly selected to exchange places. This particular operator is not a hill-climber, it is more of a mutation operator.

3.2.2 Tour Building

Unlike the above methods of moving the Swarm, these methods build the Position city-by-city. Although sometimes classified as hill-climbers, they may be better classified as “hill-jumpers”. They typically combine information from two separate Particles in an attempt to use good information to create a better individual. If the two individuals are on the same hill, the result is likely hill-climbing. If the two individuals are each at the top of their own hill, the result is hopefully a different hill that reaches even higher. The tour building operators are PSO_AS, AS, Greedy, Insert, and Random.

3.2.2.1 The PSO_AS Primary Means of Motion

This algorithm combines the ideas presented in Appendix A. As in AS (see A.8), the particles are moved in a step-by-step process to create a tour. Unlike AS, a table of

pheromone levels is not kept. Instead, choices are based on the global best particle and the local best particle as in PSO_TSP (see A.10). In fact, the algorithm is identical to that presented for PSO_TSP (see A.10.2), but the meaning behind applying the equations in step 2 alters the way the algorithm performs this step. When a Particle is in a given city, it generates a random percentage. This percentage determines if the particle attempts to follow the global best, the local best, or takes a pseudo-random path based on the particle's position before commencing a new tour. If the global best is chosen, the cities in the global best (adjacent to the city the particle is currently in) become the first choice. In the case of asymmetric problems only one choice is allowed. If the global best choices are available (i.e., they have not yet been visited in this tour), one is chosen at random and the particle proceeds to that city. If there is no available global choice or if local best is selected, the local best position is used to determine possible choices in the same way as the global best was as outlined. If the local best choices are not available, or if the "default" method is chosen, the most recently visited city (last if possible) from the Particle's current position that has not yet been visited is chosen. Other configurable "default" methods are available and explained in 3.2.4.

3.2.2.1.1 Definition of Equation 2 and Equation 3 Terms

Equation 2 and Equation 3 define PSO and require the use of the global and local best in the search process. While PSO_AS uses these equations, the following redefinition of terms is required to describe the new algorithm (see also pg 61):

- 1) V_{t+1} - The particle's new velocity for the next generation. A summation of step-by-step choices that generates a new position.
- 2) C_1 - A probability of selecting to use information of the prior tour in creating the new tour. A measure of how much the particle "trusts" its own exploration.
- 3) V_t - The particle's current velocity. A particle's velocity and position contain redundant information.
- 4) \oplus - denotes local choices made based on the probabilities. $C_1 + C_2 + C_3 = 100\%$.
- 5) C_2 - A probability of selecting to use information of the neighborhood best. A measure of how much a particle "trusts" its neighborhood best.
- 6) $P_{ig,t}$ - The neighborhood (from i to g) best position. Also known as "local" best.
- 7) $-$ - The difference of two positions is the velocity that will transform the second position into the first position. This velocity is equivalent to the first position, thus $P_{ig,t} - X_t = V_{ig,t}$.
- 8) X_t - The current position
- 9) C_3 - A probability of selecting to use information of the global best. A measure of how much a particle "trusts" its global best.
- 10) $P_{vg,t}$ - The global best position

- 11) "+" – The transformation of a position using the velocity (yields a position). Since the velocity contains redundant information with the position, this step is inherent.
- 12) X_{t+1} - The particle's new "moved" position. The position of the next generation.

3.2.2.2 Example: Moving a Particle

Let's look at an example to illustrate how PSO_AS works.

Given a TSP problem where:

$$X_t = \{2, 4, 6, 5, 3, 7, 1\}$$

$$P_{vg,t} = \{3, 7, 6, 4, 5, 1, 2\}$$

$$P_{ig,t} = \{7, 6, 2, 3, 5, 4, 1\}$$

$$C_1 = 10\%$$

$$C_2 = 10\%$$

$$C_3 = 80\%$$

$$\text{Random numbers generated} = \{60, 82, 87, 26, 94\}$$

$$X_{t+1} = \{1, 2, 3, 5, 4, 7, 6\}$$

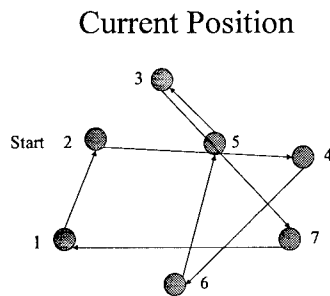


Figure 6 - X_t

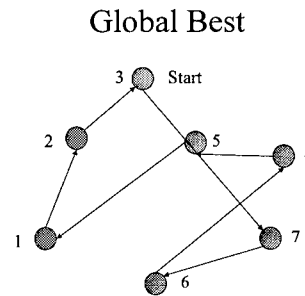


Figure 7 - $P_{vg,t}$

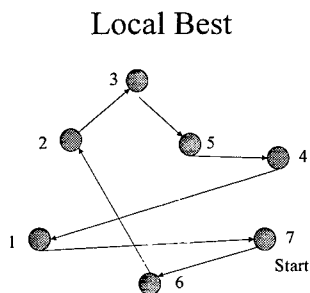


Figure 8 - $P_{ig,t}$

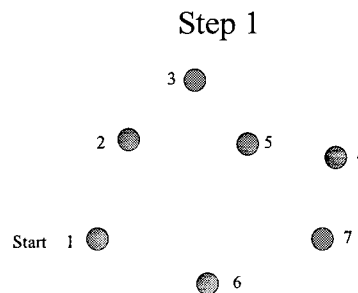


Figure 9 - Step 1

Step 1 (Figure 9): The default method is always used to select the starting city, thus city 1 is chosen since it is the last city in X_t .

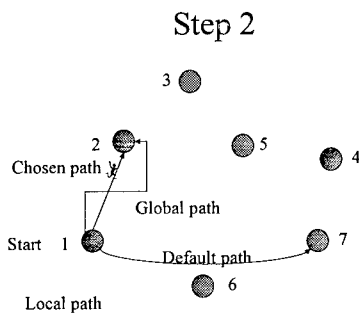


Figure 10 - Step 2

Step 2 (Figure 10): Since 60 is less than C_3 , the global information is used. City 2 is chosen since it follows city 1 in the global position. If the problem were symmetrical, city 5 may have been chosen instead since it precedes city 1.

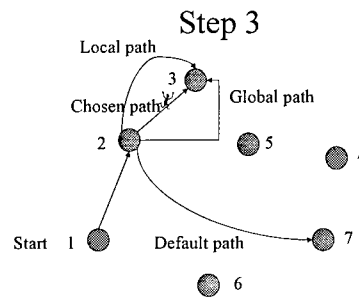


Figure 11 - Step 3

Step 3 (Figure 11): Since 82 is greater than C_3 but less than $C_3 + C_2$, the local information is used. City 3 is chosen since it follows city 2 in the local position. When both the global and local positions contain the same paths, the probability of continuing to follow that path is equal to $C_3 + C_2$.

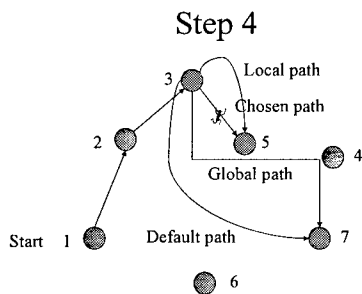


Figure 12 - Step 4

Step 4 (Figure 12): Since 87 is greater than C_3 but less than $C_3 + C_2$, the local information is used. City 5 is chosen since it follows city 3 in the local position.

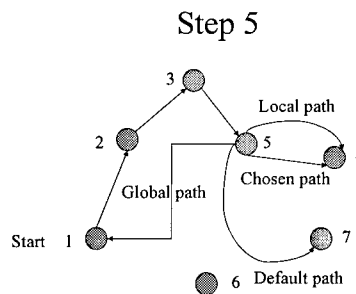


Figure 13 - Step 5

Step 5 (Figure 13): Since 26 is less than C_3 , the global information is used. City 1 follows city 5 in the global, but cannot be chosen since it has already been included in the tour (Step 1). We therefore attempt to use the local information. City 4 is chosen since it follows city 5 in the local position. If a local position's information cannot be used

because it has already been included in the tour, no attempt is made to use the global information. Rather, the default information is used. When local information is used, the probability of continuing the path when the global information has already been included in the tour is equal to $C_3 + C_2$, thus greatly increasing the chances of using the local information.

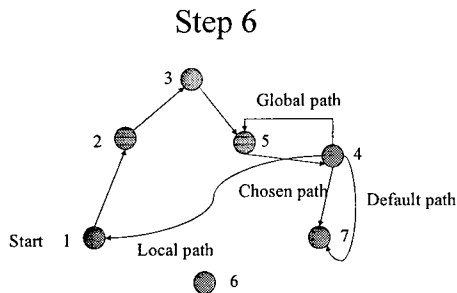


Figure 14 - Step 6

Step 6 (Figure 14): Since 94 is greater than $C_3 + C_2$, the prior tour information or default is used. City 7 is chosen since it is the last in the list that has not already been included in the tour. City 1 was included in Step 1.

Step 7 (Figure 15): Since city 6 is the only city which has not been included in the tour, it is selected.

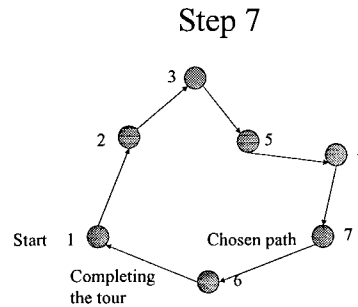


Figure 15 - Step 7; X_{t+1}

While the example demonstrates finding the global optimal solution, such is not necessarily the case. The new tour is built from Particles that have survived, so the new tour is also likely to be good.

3.2.2.3 Ant System (see A.8)

This was actually the first means of moving the Swarm written into code, but poor performance drove the development of other methods. Under this method, the “Velocity” of each Particle is a table analogous to a pheromone table. A tour is constructed by first randomly choosing either to follow the global best, the local best, or the current Particle. Then, using the selected Velocity, probabilities for visiting cities not added in the tour are generated in a “roulette” method. Paths that have higher Velocity will also have higher probability of being selected. After tours have been constructed, those Particles with fitness values greater than or equal to the median are “rewarded” and adjust their

respective Velocities by increasing those values along the links in the tour. Those with fitness values less than the median are “punished” and decrease those values along the links in the tour.

Preliminary testing show this method to be rather slow and that it does not converge to a good quality of solution very well. Thus, to improve convergence, a single “global” Velocity replaced the individual Velocities. It contains the combined information, thus is updated as above for each Particle. Under this variation, all “Velocities” are identical, so tours are created using only one Velocity. The code supports both variants. See A.8 for details on AS.

3.2.2.4 Greedy

Starting with the last city in the current tour, perform the greedy algorithm (see A.1).

3.2.2.5 Insert

Perform the insert algorithm (see A.2) using the current tour as the order of selection of cities instead of randomly selecting cities.

3.2.2.6 Random

Randomly select a city not in the current tour and add it to the tour. Continue until all cities are added.

3.2.3 *Random Choice of Method*

Not to be confused with random, this option randomly selects for each Particle one of the above permutation or tour-building methods excluding AS. The book-keeping involved with updating the Velocity table (regardless of operator choice) made AS unwieldy to include.

3.2.4 *The Default Selection Method in the PSO_AS Operator*

While the selection method explained in the example (see 3.2.2.2) describes defaulting to the last city available that was visited in the prior tour, there are many other configurable options. The PSO_AS operator is tour building, so greedy, insert, random, and random choice are also available as default methods. AS is not available as a default method due to the inefficiency of maintaining the Velocity discussed above. Random

choice, in this case, selects one of the other available default methods every time the default method is used.

3.3 Initialization Methods

Of course, before a Particle ever moves it must be initialized. The initialization method is configurable for the Swarm (all Particles are initialized with the same method). Because of good design, the code to initialize them is the same as that to move them. The only difference is the hill-climbing portions which continuously perform the routine until no more improvement can be realized, thus completely agreeing with the appropriate algorithm. In other words, the “outer-loop” is also performed (see 3.2.1 regarding outer loop on pg 14). The available initialization methods are random (see 3.2.2.6), insert (see 3.2.2.5), re-insert (see 3.2.1.2), greedy (see 3.2.2.4), city-swapping (see 3.2.1.1), 2 or 3-Opt (see 3.2.1.3), Lin-Kernighan (see 3.2.1.5), and random choice (see 3.2.3).

3.4 Hill-Climbing after Moving

After moving the Swarm but before evaluating, it may be profitable to perform some hill-climbing. With efficient design, this is simply just moving the Swarm again. With the exclusion of AS, all methods of moving the Swarm (see 3.2) are available for hill-climbing after moving. While it is intended that those algorithms designated as hill-climbing be executed at this point, any of the options (or none) may be used.

3.5 Re-hope methods

If the Swarm has moved without any change in the global best Position for a (configurable) number of iterations, we assume that the Swarm has “converged”. This step is used to allow the Swarm to explore even more. Approaches include hill-climbing, re-initialization, and dynamic parameter adjusting.

The hill-climbing approach attempts to find a superior solution by performing one of the hill-climbing algorithms discussed previously. Such hill-climbing is done either on the entire Swarm, or on just the global best Position. If a superior solution is found, moving the Swarm searches a different local area since it now has new information.

Re-initialization affects all particles with the exception of the global best. After re-initializing, searching is commenced anew. The goal of this approach is to let the

particles find new local best solutions, and possibly in the process find a new global best as they re-converge.

Dynamic parameter adjusting alters the parameters to encourage more exploration or more exploitation. For example, if the percentage for global best (C3) is 95%, we might be doing too much exploitation and not enough exploration. This percentage could be lowered to 90% (with perhaps the default (C1) gaining 5%). Now, as the search continues there will be more exploration. This method can be applied many times. For every 50 iterations a new solution is not found, the parameter could be adjusted another 5% until a threshold is reached (say 60%). If, on the other hand, a new solution is reached, the parameters are reset to their original values. Many parameters including Swarm moving methods and hill-climbing methods can be adjusted. With the exception of parameter adjusting, Re-hope methods are again moving the Swarm (see 3.2) and all of the same options may be used.

Re-hope methods are also nested. For example, perform parameter adjusting until the threshold is reached, then perform hill-climbing (and reset parameters). If hill-climbing fails, try re-initializing. If searching after re-initializing fails . . . the program ends.

3.6 Summary

PSO_AS is functionally complex, yet structurally simplistic. It has the same high-level structure as any PSO algorithm: (1) the Swarm is initialized (and initially evaluated), (2) followed by hill-climbing and evaluation, (3) if progress is not seen after several iterations, some Re-hope method(s) are used and the algorithm continues per steps (2), (4) After all re-hope methods are exhausted, the Swarm exits and reports results.

While it inherits the structure from PSO, it differs in that the manner of moving the Swarm. PSO_AS uses a tour-building process rather than a permutation (see 3.2.2.1). Even though this tour building process was inspired by AS, it is nevertheless also a valid feature of PSO because it uses the information gleaned from prior successful Particles (the global and local best) as a meta-heuristic to guide the building process rather than the pheromone tables of AS.

4 Design of Experiments

The goal of this thesis is to produce a better algorithm and this section is concerned with tests that compare PSO_AS to an existing algorithm (or “enhancements” to PSO_AS such as parameter tuning). What does the term “better” mean and how do we know if the goal is met? There are several ways an algorithm can be “better”. The primary metrics of performance are quality of solution (effectiveness), and speed (efficiency) [Barr], however it is rare that an algorithm always is more effective and efficient for all classification of problems (robustness) [Wolpert (NFL)]. Thus, a well-designed experiment must gather data for a variety of problem classes. The questions we are attempting to answer are: 1) How good is the final solution? 2) How fast did it arrive at the final solution? 3) Does it perform consistently on a wide range of problems? 4) How does PSO_AS compare to other algorithms?

We start with considerations that apply generally to all tests conducted such as hardware used, test problems used, what data is collected and how the data will be used to answer the questions regarding effectiveness, efficiency, and robustness. We then move to specific tests conducted.

4.1 General Testing Considerations

Consistency reduces variables. Although this is perhaps obvious, it nevertheless is the reason why as many tests in this chapter as possible share these common features. The only exceptions to these common testing considerations are those tests conducted for the purpose of comparison to published results (see 4.2.2.13 and 4.2.2.14). In such cases, the tests are designed to meet the same criteria as that used to produce the published results.

4.1.1 TSP Standard Test Suite

[TSPLIB] is a widely used suite of problems specifically for the TSP. As such, it allows comparison between published results. In fact, some of the tests conducted are performed on the same test problems as that used to produce the published results. It is comprised of many problems with various numbers of cities, the solutions to which are known. It includes both symmetric and asymmetric problems.

Another common testing method (not used in this thesis) is random generation of problems. While this method presumably creates problems without specific bias (as long as several are created and tested), its problems include a lack of known solution and the unavailability of the problem for other researchers.

4.1.2 Complexity and Classification of TSP Problem

TSP is $O(n!)$, hence problem complexity increases with the number of cities. Tests are conducted with four problems from [TSPLIB] in the following classifications: 1) Symmetric Small (100 –200 cities), 2) Symmetric Medium (201-500 cities), 3) Symmetric Large (501-1000 cities), 4) Symmetric Extra Large (>1001 cities), 5) Asymmetric Small (50-200 cities), and 6) Asymmetric Large (>200 cities). Problem sizes smaller than these are easily and quickly solved. Also, since asymmetric problems are more complex than symmetric problems, asymmetric problems of greater than 600 cities do not exist in [TSPLIB]. In fact, the asymmetric problems selected are the eight largest problems in the library and the only ones that fit the category. The individual categories were chosen arbitrarily, but with the intent to adequately represent the problems contained in [TSPLIB].

4.1.3 Collection of Data

PSO_AS is compared to PSO_TSP, Lin-Kernighan, AS, and Inver-Over. Therefore, ten runs (for statistical accuracy) for each algorithm are executed for each problem in each of the symmetric classifications. These tests are performed to obtain performance baseline data used for qualitative analysis.

PSO_AS also has several parameters (how much to “trust” local, global or default method, and what default method to use) which must be “tuned” or adjusted to find the best parameters for quality of solution, speed, and robustness. For these tests, three (smaller) symmetric TSP problems (d198.tsp, si535.tsp, u724.tsp) are executed for ten runs with each varying tunable parameter. The sheer numbers of tests required for tuning prohibits the use of a wide variety of test problems. By keeping the problem set and size small, each test completed within a four day period.

4.1.4 Data Reduction/Presentation

Raw data collected for each run of all tests is the fitness value of the best member in the population (Swarm), the time required (after problem loading and initialization) to find the best fitness value, and the total number of iterations (i.e., how many times we moved the Swarm) before termination. The raw data represents the results of over 2000 individual runs of the algorithm and required over 1000 CPU run-time hours to collect. To make it comprehensible, average percentage away from optimal solution and average time to solution for all runs of a problem is calculated and charted. This gives a graphical view of algorithm performance for each test. Associated standard deviations are also calculated for all problems (contained in spreadsheets) and displayed for algorithm comparison purposes. Use of standard deviation, median, maximum or minimum information when comparing two algorithms with unknown distribution gives inaccurate results. They are included and displayed for “intuitive” visual purposes only. When two algorithms are compared, a [Kruskal-Wallis Test] (which does not assume a normal distribution) is performed for hypothesis testing.

4.2 Individual Test Specifics

Testing is performed either to obtain baseline data, perform parameter tuning of PSO_AS, or compare the tuned operation of PSO_AS to baseline data or published results. Since we desire to know how PSO_AS compares with other algorithms, we perform baselines of each algorithm (PSO_AS, LK, AS, Inver_Over). In these baselines, we use consistent Swarm size and ending criteria. We also tune PSO_AS for optimal performance with respect to quality of solution and speed of solution. Finally, we perform runs that demonstrate the operation of PSO_AS after it has been tuned. All test-script details (including settable parameters not discussed) are found in [Appendix C - Testing Scripts].

4.2.1 Baseline Data

Since we want to know how PSO_AS compares with other algorithms, we perform baselines of each algorithm, (LK, PSO_AS, AS, Inver_Over). In these various baselines, the Swarm size and the ending criteria are the same.

4.2.1.1 Experiment 1 - LK Baseline

LK (see A.7) is the most effective heuristic algorithm currently known for TSP [Lawler]. It is a hill-climber in the sense that it is a many-to-one mapping of all possible candidate solutions to a subset of all possible candidate solutions. It also has the desirable property that the fitness value of the mapped candidate is guaranteed to be strictly lower than all other candidates that map to it. Thus, it is an ideal hill-climbing operation.

It is also deterministic. Performance of LK is often demonstrated by taking a single potential candidate (usually the tour {1,2,3,4, . . .}) and executing LK while measuring the time required to produce the solution. The quality of this single data point is then used to characterize the algorithm. Such analysis and comparison between LK and a non-deterministic algorithm produces claims such as “Lin-Kernighan algorithm takes a fraction of time necessary for our algorithm. However, the precision of results is much lower . . . If Lin-Kernighan algorithm was run for the same time as our evolutionary system, probably it would win the competition easily. [Tao]”

In order to produce a competition where the results can be fairly compared, we perform a baseline on a version of LK that is given the same stopping criteria as PSO_AS and all other baselines as well. For each generation, 100 randomly generated candidates are improved by LK. The lowest fitness value of the current generation is compared to the lowest fitness value of all prior generations. If the new generation produces a lower fitness value, this value and the accumulated CPU time required to produce the value is retained. When six consecutive generations have occurred without any improvement in fitness value, the program reports results of best fitness value, CPU time used at the time of discovering the best solution, and total number of generations (including the six without any improvement).

4.2.1.2 Experiment 2 - PSO_AS Baseline

The PSO_AS baseline has a Swarm size of 100 and a neighborhood size of ten. It “trusts” the local and global best solutions 10% each, and the default method 80%. It uses the default method discussed in (3.2.2.2). It initializes the Swarm using LK, and performs LK on all and the Positions in the Swarm after the Swarm has been moved (see Moving the Swarm). The Swarm size and neighborhood size are based on [7.3.3], the use of LK is discussed in [Appendix B], and the “trust” values are selected arbitrarily. Tuning the algorithm focuses on finding good “trust” values and examines the Greedy default method (see 3.2.2.4 and 3.2.4). Ending criteria is discussed in 4.2.1.1.

4.2.1.3 Experiment 3 – AS Baseline

The AS baseline has a Swarm size of 100 and uses the same stopping criteria as other baselines (see 4.2.1.1). Two baselines are produced. In one, the algorithm works as described in Appendix A.8. In the other, each “ant” is optimized after each generation and all are rewarded based on the assumption that all LK optimized tours are good.

4.2.1.4 Experiment 4 – Inver-Over Baseline

Inver-Over uses a population size of 100 and a 2% Over amount as per [Tao]. It likewise uses the same stopping criteria as other baselines (see 4.2.1.1).

4.2.2 PSO_AS Parameter Tuning Tests

We also need to tune PSO_AS for optimal performance with respect to quality of solution and speed of solution before we can compare PSO_AS to other baselines. These tests each use the parameters discussed in the baseline as a starting point (see 4.2.1.2) and vary a single parameter (or two in the case of percentages where lowering a value means raising another to maintain 100%) in order to determine the effects of varying the parameter.

4.2.2.1 Experiment 5 – Course Tuning PSO_AS: 10% Default “Trust”

By keeping the default trust at 10% and varying the global and local trust in 10% increments $\{(90,0),(80,10),(70,20), \dots\}$ we compare global and local trust. All other parameters are the same as the baseline (see 4.2.1.2).

4.2.2.2 Experiment 6 – Course Tuning PSO_AS: 10% Local “Trust”

By keeping the local trust at 10% and varying the global and default trust in 10% increments $\{(90,0),(80,10),(70,20), \dots\}$ we compare global and default trust. All other parameters are the same as the baseline (see 4.2.1.2).

4.2.2.3 Experiment 7 – Greedy PSO_AS: 10% Default “Trust”

This experiment is the same as 4.2.2.1 with the exception of using the Greedy default method to determine the effects of using the Greedy method.

4.2.2.4 Experiment 8 – Greedy PSO_AS: 10% Local “Trust”

This experiment is the same as 4.2.2.2 with the exception of using the Greedy default method to determine the effects of using the Greedy method.

4.2.2.5 Experiment 9 – Fine Tuning PSO_AS: 90% Global “Trust”

By keeping the global trust at 90% and varying the local and default trust in 1% increments $\{(90,10,0),(90,9,1),(90,8,2), \dots\}$ we compare local and default trust. The 90% value for global trust is based on experiments 5 and 6. All other parameters are the same as the baseline (see 4.2.1.2).

4.2.2.6 Experiment 10 – Fine Tuning PSO_AS: 85,95% Global “Trust”

This is the same as 4.2.2.5, but varying the values for global trust in order to see how fine-tuning adjustments to this parameter affects performance.

4.2.2.7 Experiment 11 - Greedy PSO_AS: 85,90,95 Global “Trust”

This is the same as 4.2.2.6, but using the Greedy default method.

4.2.2.8 Experiment 12 – PSO_AS Tuned for Quality of Solution

Using prior results, the baseline PSO_AS is tuned with 2% default, 8% local, 90% global “trust”. The Greedy default method is used. This “tuned” version is then used for comparison against other algorithms.

4.2.2.9 Experiment 13 – PSO_AS Tuned for Speed

Using prior results, the baseline PSO_AS is tuned with 2% default, 8% local, 90% global “trust”. This “tuned” version is then used for comparison against other algorithms.

4.2.2.10 Experiment 14 – PSO_AS Tuned for Robustness

Using prior results, the baseline PSO_AS is tuned with 10% default, 45% local, 45% global “trust”. This “tuned” version is then used for comparison against other algorithms.

4.2.2.11 Experiment 15 – Parallelization of PSO_TSP

While this experiment may seem out of place here, it is of historical significance. The results of this experiment drove the creation of PSO_AS. Rather than describe the experiment here, a full unpublished paper is included in Appendix E .

4.2.2.12 Experiment 16 – PSO_TSP vs PSO_AS

While not strictly a comparison between the two algorithms, the results do support the claim that PSO_AS is superior. Rather than describe the experiment here, a full paper accepted for publication is included in Appendix F.

4.2.2.13 Experiment 17 –Tuned PSO_AS vs Published Results of Inver-Over

The tuned for quality of solution PSO_AS is executed with the same stopping criteria as that published by [Tao]. Instead of six generations without an improvement, we use ten generations without an improvement. We also use all problems of greater than 100 cities with published results that are in [TSPLIB] (there were four). In this way, we compare the result (quality of solution only) with the published results of Inver-Over.

4.2.2.14 Experiment 18 –Tuned PSO_AS vs AFIT Research

The tuned for quality of solution PSO_AS executed as per 4.2.2.8 on standard problems used to test Reactive Tabu [Harder, Kinney, O'Rourke]. It is also executed vs the same problems used to test [Hall].

4.3 Summary of Tests

The following table helps summarize the tests:

Test	Purpose
1	Baseline of LK used in comparison analysis against PSO_AS.
2	Baseline of PSO_AS used for a starting point in the parameter tuning process.
3	Baseline of AS used in comparison analysis against PSO_AS.
4	Baseline of Inver-Over used in comparison analysis against PSO_AS.
5	Course comparison of varying global vs local trust in 10% increments. Default value maintained constant at 10%

6	Course comparison of varying global vs default trust in 10% increments. Local value maintained constant at 10%.
7	Course comparison of varying global vs local trust in 10% increments while using the Greedy default method. Observe the effects of using the Greedy default method.
8	Course comparison of varying global vs default trust in 10% increments while using the Greedy default method. Observe the effects of using the Greedy default method.
9	Fine comparison of varying local vs default trust in 1% increments. Global value maintained constant at 90%.
10	Compare Global values 85% and 95% to data obtained for Global value of 90%.
11	Compare fine-tuned Global values using the Greedy method.
12	Collect performance data for PSO_AS tuned for quality of solution. This data is then used for comparison against other baseline data.
13	Collect performance data for PSO_AS tuned for speed. This data is then used for comparison against other baseline data.
14	Collect performance data for PSO_AS tuned for robustness. This data is then used for comparison against other baseline data.
15	Observe performance of Parallel version of PSO_TSP
16	Observe difference in PSO_TSP and PSO_AS
17	Compare PSO_AS with published results of Inver-Over
18	Compare PSO_AS with published AFIT results.

5 Results and Analysis

This chapter presents condensed results of experiments conducted. We strive to assert that PSO_AS is superior to LK, AS, Inver-Over, and other published results. It follows the same logical progression of experiments outlined in the previous chapter (baselines of algorithms, tuning of PSO_AS, Comparison of PSO_AS to other algorithms), but groups them together when a visual representation of combined results is meaningful.

5.1 Experiment1 - LK Baseline, Experiment 2 - PSO_AS Baseline

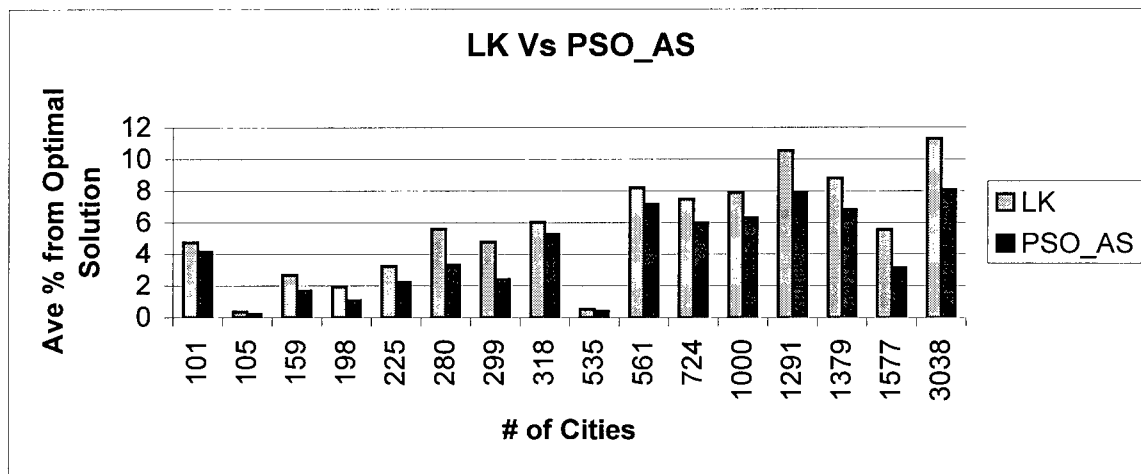


Figure 16 - PSO_AS and LK Baseline Solutions

Surprisingly, the baseline PSO_AS shown in Figure 16 produced better average solutions in all cases when compared to LK. For example, at 561 cities, LK has an average of 8.22 whereas PSO_AS has an average of 7.15. While this is certainly encouraging, it was unexpected. With such a high percentage of default method (80%), the expectation was for poorer results. With the high default, large numbers of the current Position are selected. This results in sections of the current Position being inverted and added to the new Position. The inversion of large portions of a successful Position account for the good results.

Detailed information such as error bars depicting standard deviations is not represented at this time since the PSO_AS baseline will not be used for comparison purposes. Such detail will be added later in this chapter.

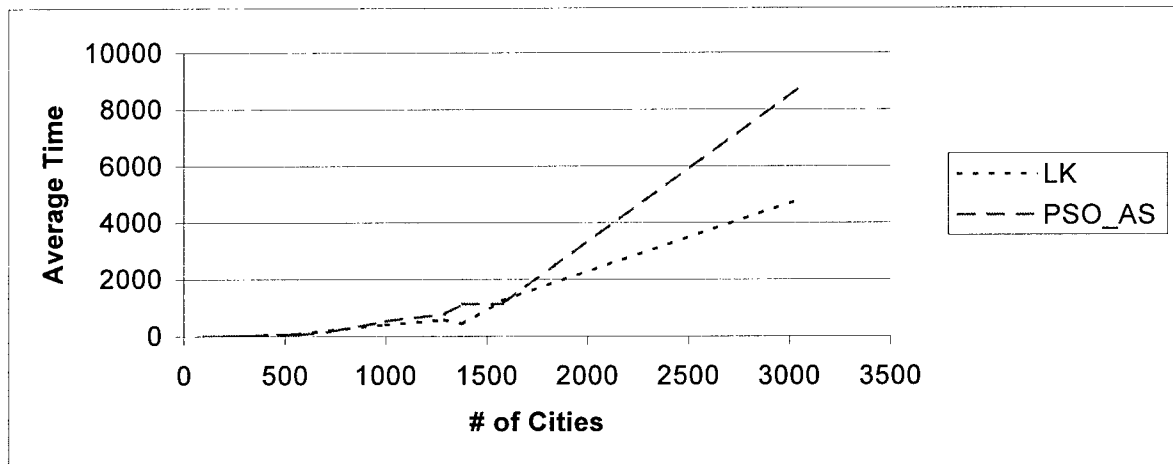


Figure 17 - PSO_AS and LK Baseline Average Time (sec)

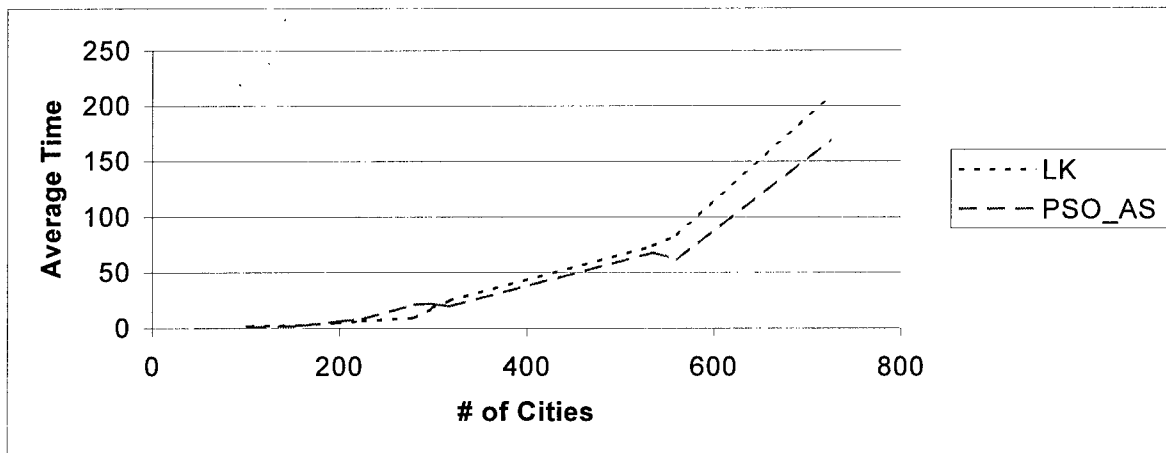


Figure 18 - PSO_AS and LK Baseline Average Time (sec): Cities < 800

Figure 18 represents the same data as that in Figure 17, but scaling is altered to allow closer inspection when the number of cities is less than 800. PSO_AS produced solutions in better time in nine of the sixteen cases. In all cases, the time per generation is smaller for PSO_AS, but more time is spent productively finding better solutions (i.e., it takes more generations to meet the stopping criteria of the test). While this result was anticipated, it is counter-intuitive. Inspection of the algorithm indicates the only difference between the two runs is PSO_AS must do more work during the tour-building

step. Both generate the same amount of random numbers (LK to produce the random Position, and PSO_AS to choose which “best” to follow). Both perform LK on each Position. So how can PSO_AS be faster? The savings is realized in the LK portion of the algorithm. The Position built by PSO_AS is closer to optimal than that produced randomly. Because of this, less work needs to be done to reach the local LK optimal Position.

While these results are encouraging, it is expected the tuned version shall have even better results. Comparison analysis follows the tuned tests.

5.2 Experiment 3 – AS Baseline

The AS baseline produces poor results, with solutions typically greater than 200% of the optimal solution. It takes many more ants to lay down a pheromone trail than was allowed by this experiment. In an attempt to “fix” this, the algorithm was altered to perform LK on each Position (as PSO_AS does) and lay pheromone for every Position (on the assumption that all LK produced Positions are “good” and should be rewarded). The raw data results are in Appendix D.2 and show that AS produces solutions nearly the same as LK, but requires twice as much time.

5.3 Experiment 4 – Inver-Over Baseline

We attempted to obtain original source code for this algorithm. Unfortunately, the author could not be reached. We wrote our own source code based on the paper by [Tao]. Although the paper is plainly written, the resultant version does not re-produce the results claimed, and the results are quite inferior to LK. Subsequently, we were able to contact the author and have a version (not the exact one used to produce the paper) that “should implement the algorithm correctly.” It likewise produced results similar to the code we wrote. The raw data results are in Appendix D.3.

5.4 Experiment 5 – Course Tuning PSO_AS: 10% Default “Trust”, Experiment 6 –
Course Tuning PSO_AS: 10% Local “Trust”

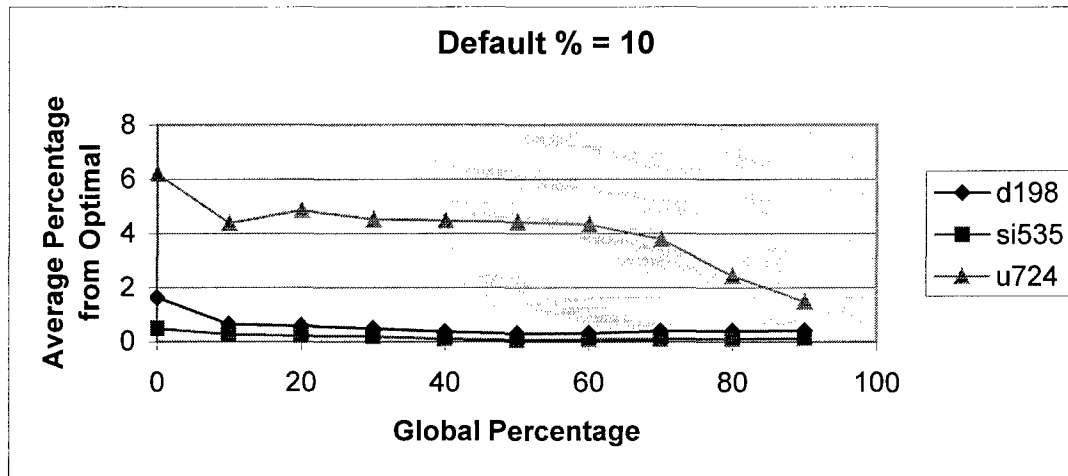


Figure 19 - Quality of Solution vs Course Global Percentage (Default = 10%)

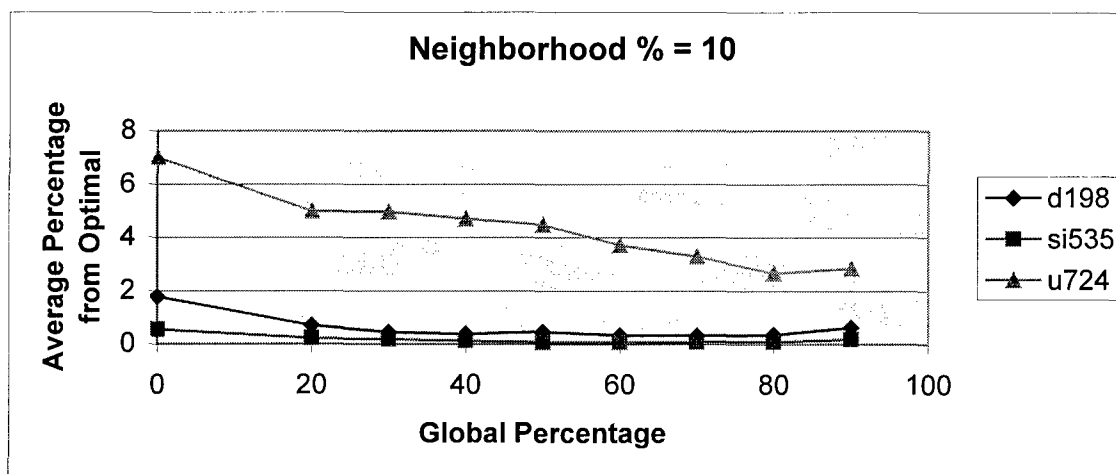


Figure 20 - Quality of Solution vs Course Global Percentage (Neighborhood = 10%)

The observed trend matches the expected result that a higher global percentage gives a better quality of solution.

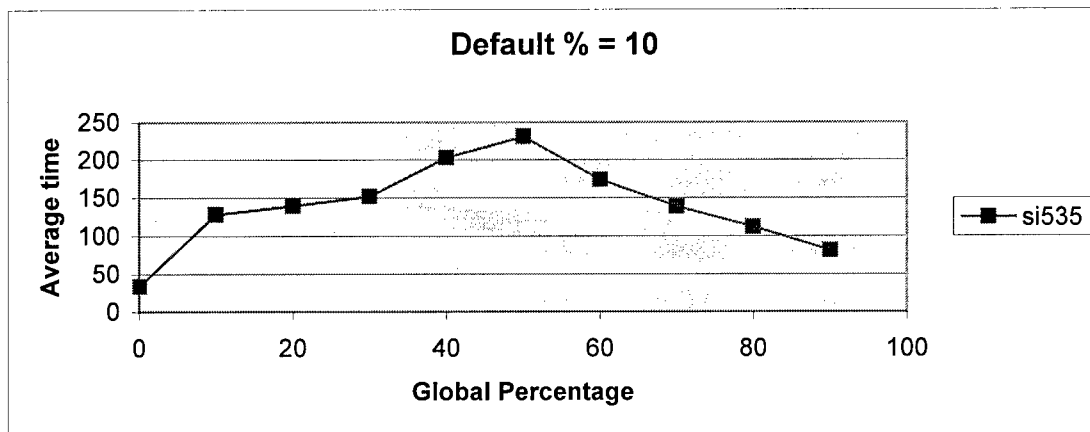
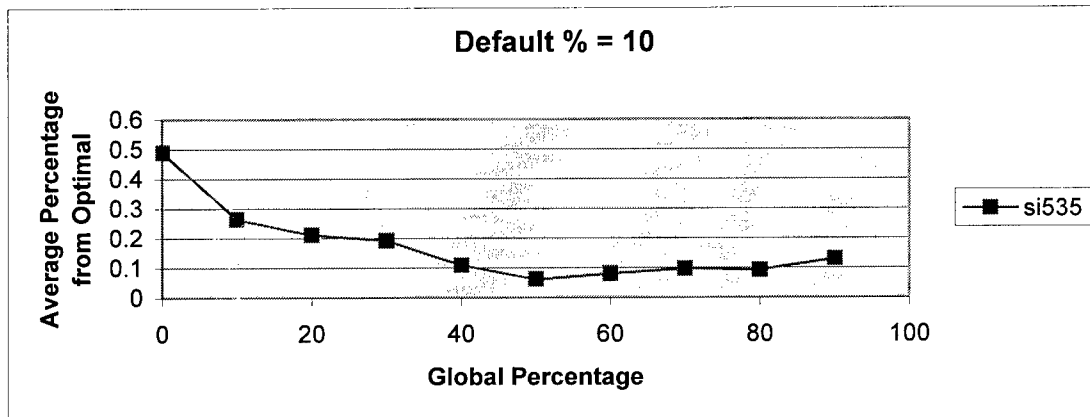


Figure 21 - Average Percentage from Optimal and Average Time (sec) vs Course Global Percentage (Default = 10%)

The results for si535.tsp are typical of all experiments conducted. In general, the shape of the time required curve is a mirror image of the percentage from optimal curve. This is not surprising since it takes more time to find a better solution.

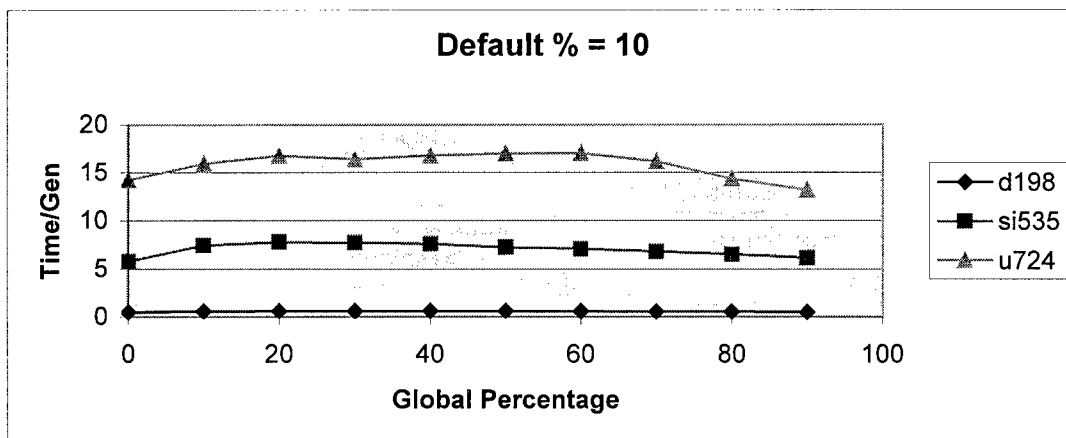


Figure 22 - Time/Generation (sec) vs Course Global Percentage (Default = 10%)

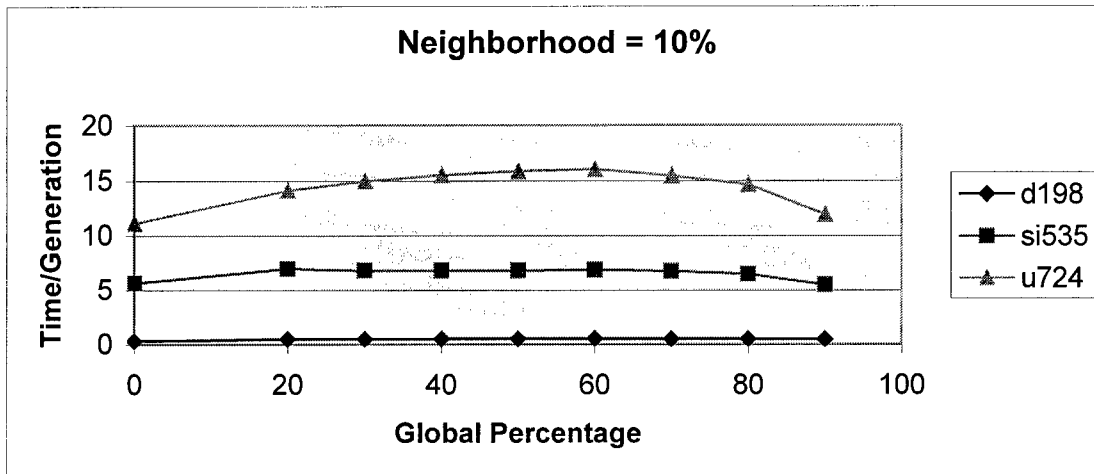


Figure 23 - Time/Generation (sec) vs Course Global Percentage (Neighborhood = 10%)

The curves are convex down, with lower values at the endpoints and higher values in the middle. The lower values represent a measure of efficiency. Thus, good solutions (see Figure 19, where the better solutions are produced with high global percentage) are produced most efficiently with a high global percentage.

5.5 Experiment 7 – Greedy PSO_AS: 10% Default “Trust”

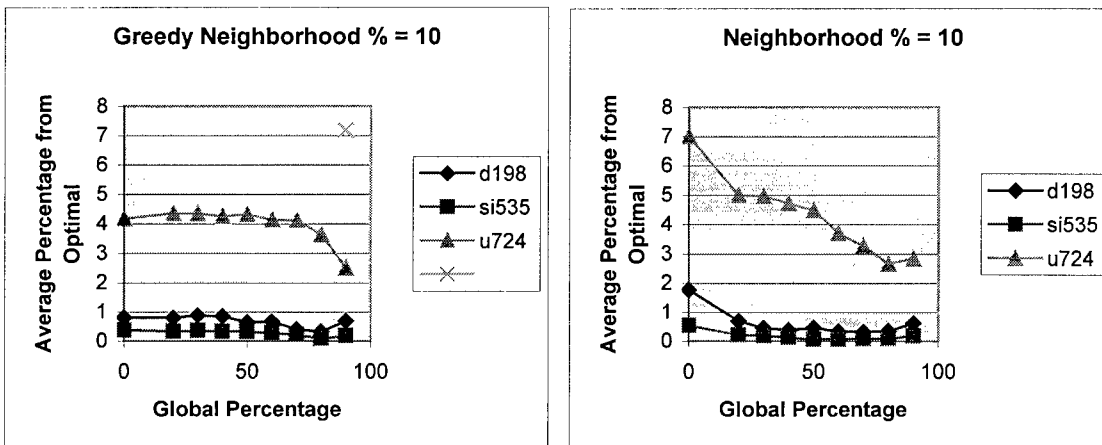


Figure 24 - Greedy Quality of Solution (Default = 10%)

In general, it appears that the greedy default method produces superior solutions to non-greedy. An examination of u724.tsp demonstrates some significant exceptions. When the global percentage is high, the greedy method produces poorer results. When the global percentage is near 90%, it is unclear which is better. Further testing is done (see 5.7) to more closely examine the greedy method with a global value near 90%.

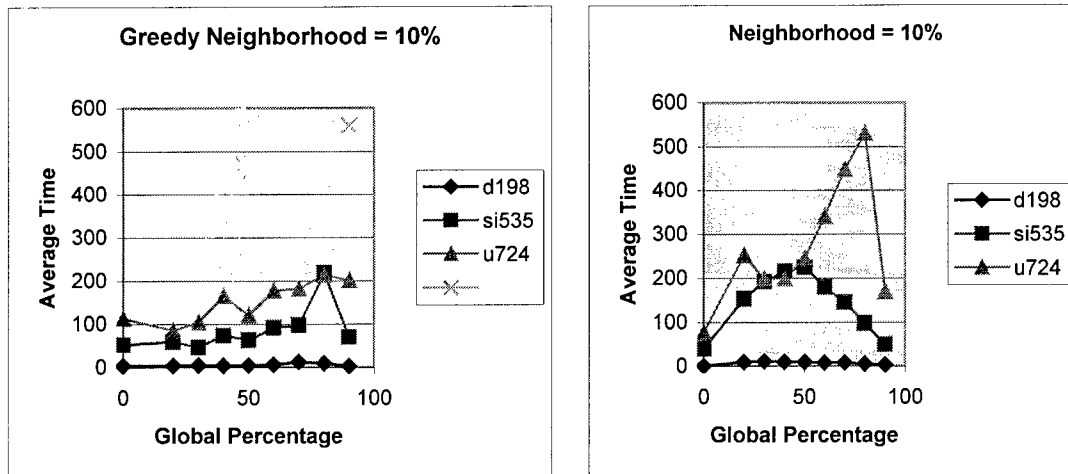


Figure 25 - Greedy Time (sec) vs Global Percentage

Once again, the general case is that the greedy default method is superior and produces solutions faster as seen by the u724.tsp and si353.tst curves on the left being lower than their associated curves on the right.

These results are significant with respect to robustness. The “flatness” of the greedy curves indicates that with a global percentage near the middle (45%) consistently good results are obtained rather swiftly. Unfortunately, prior results indicate we desire a high amount of global percentage for best quality of solution and speed of solution. At the 80-90% global range, results (for both quality and speed) are inconclusive, thus requiring more fine-grained testing.

5.6 Experiment 8 – Greedy PSO_AS: 10% Local “Trust”

Raw data is contained in Appendix D.6. Due to the inconclusive results of Experiment 7 (requiring more testing), analysis of this data is not presented in lieu of presenting the fine-grained analysis in 5.7.

5.7 Experiment 9, 10, 11, and 12 – Fine Tuning PSO_AS: 85,90,95% Global “Trust”

With or Without Greedy Method

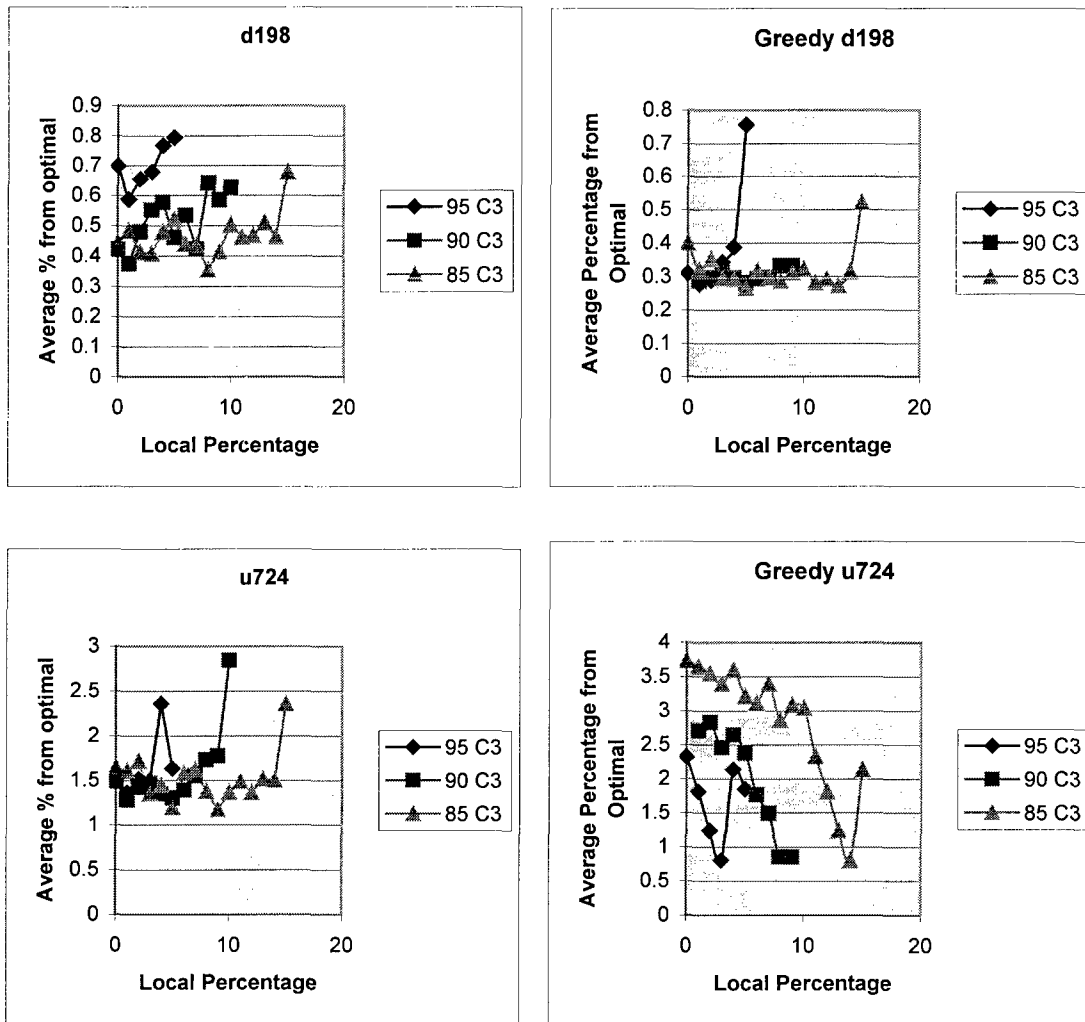


Figure 26 - Quality of Solution: Fine Tuning Global Trust

Although still not conclusive, we see that without greedy a global value of 85-90% works best with a value of 95% producing poorer quality of solutions (note how the 95 phi curve is higher than the other curves for d198,tsp). With greedy we observe that the curves for 85-95% all produce the same result dependant on the value for default “trust” (note how the curves all seem identical, but just “shifted over” based on the value of phi). It is further observed that a value of 2% for default trust produces the best quality of solution for these tests (note the minimum of greedy u724 at 95 phi is when local percentage is 3%, hence default percentage is 2%). The sharpness of the u724 curves

suggests this generalization may not hold in all cases, since other curves may be similarly shaped but with differing minimum values. We also observe in all cases that greedy produces superior quality solution (for at least a single point).

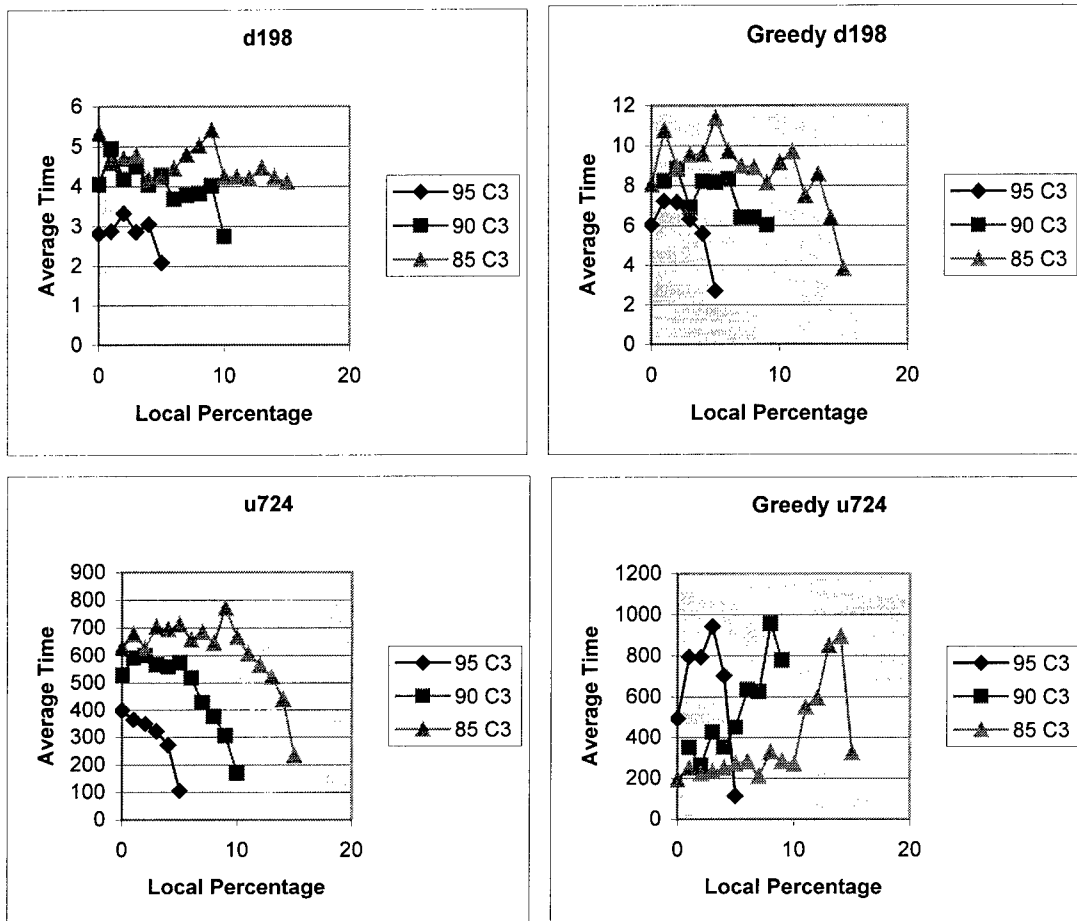


Figure 27 - Time: Fine Tuning Global Trust

Once again we see a strong relationship between the time required and the quality of solution. Time curves are nearly mirrors of quality of solution curves. We also note that in general, greedy takes less time, but at the point where the best solution is found it takes more time. For non-greedy we see the time lower as global percentage increases. This is expected since quality of solution is likewise poorer.

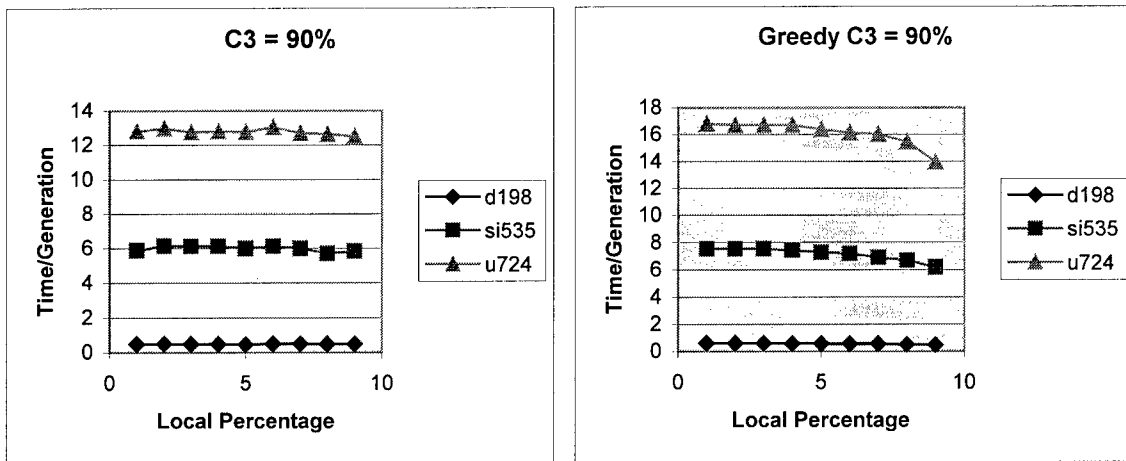


Figure 28 - Time(sec)/Generation: Fine Tuning Global Trust

For Greedy, time/generation is always higher than that for non-greedy. This suggests that non-greedy is more time efficient.

5.8 Experiment 12 – PSO_AS Tuned for Quality of Solution

In summary of prior results, best quality of solution is obtained by setting the global percentage high (85-95%) and using the greedy default method set to 2%. While the range of global percentage does not affect quality of solution, it does affect average time. Unfortunately, those results are inconclusive. We therefore select a global percentage of 90% to obtain medium speed performance.

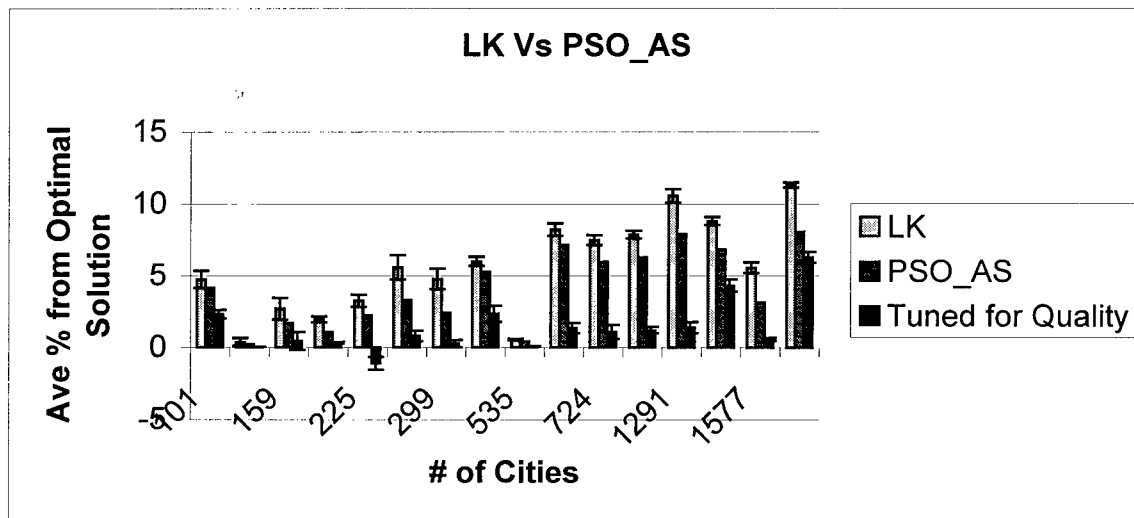


Figure 29 - LK Vs PSO_AS Quality Tuned: Quality of Solution Comparison

With PSO_AS tuned, we get better solutions in all cases when compared to the non-tuned version. Error bars represent one standard deviation from the average and are displayed for LK and the tuned version.

The negative value at 225 cities is due to rounding. Optimal solutions published in TSPLIB for Euclidean problems are found by rounding the distance between all cities. The values reported are unrounded. This experimental data point brought rounding to light and the code was written prior to this knowledge.

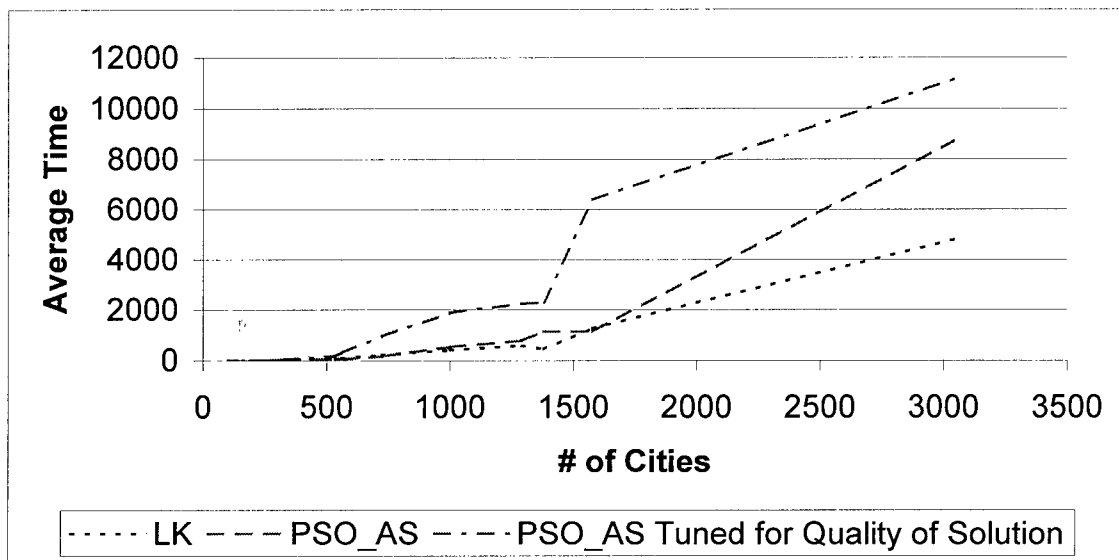


Figure 30 - LK Vs PSO_AS Quality Tuned: Time(sec)

As noted previously, obtaining better solutions requires more time.

5.9 Experiment 13 – PSO_AS Tuned for Speed

In summary of prior results, speed of solution curves mirror quality of solution curves. In order to attempt to achieve a balance between quality of solution and speed, we looked at average time per generation and observed that the non-greedy default method gives lower values. Therefore, to obtain good quality of solution with more efficiency we use the same values as that in Experiment 12, but without using the greedy method.

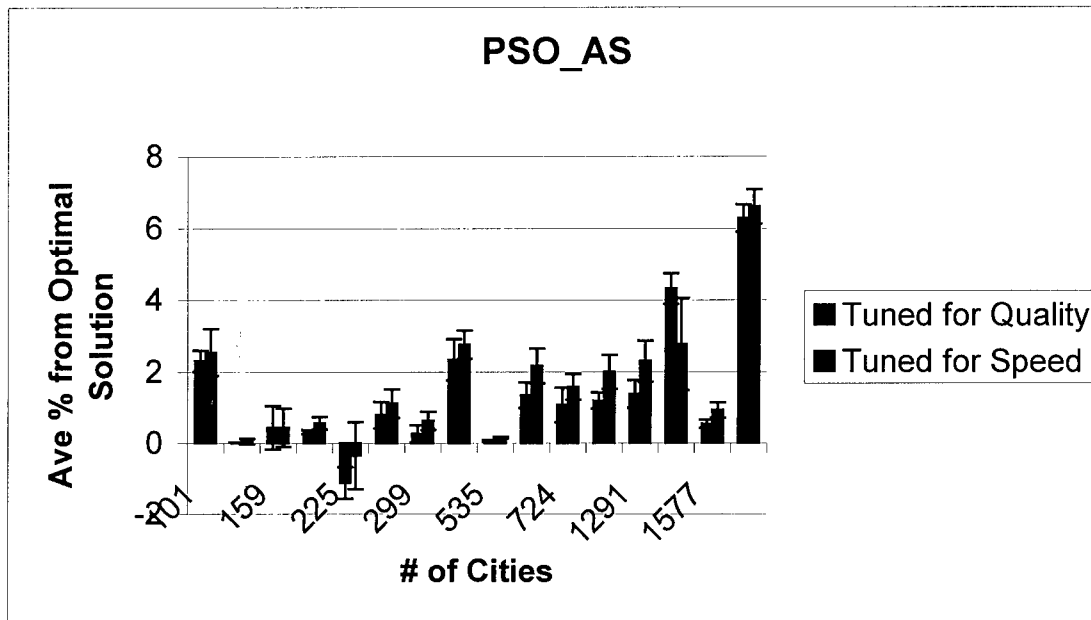


Figure 31 - PSO_AS Quality Vs PSO_AS Speed: Quality of Solution Comparison

In general, tuning for speed slightly degraded quality of solution. Note that error bars in all cases overlap suggesting (assuming the data is normally distributed) the results may not be statistically significant.

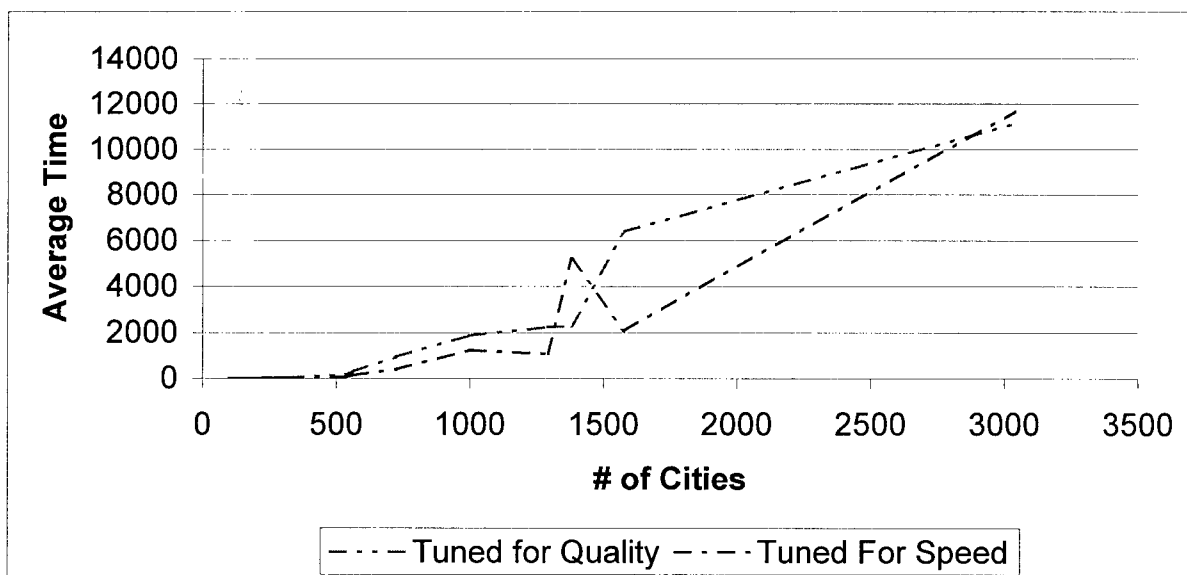


Figure 32 - PSO_AS Speed Vs PSO_AS Quality Tuned: Time(sec)

In general, tuning for speed does produce faster results as observed by the lower curve. Unfortunately, this result is not universal. In places where the speed was slower, the quality of solution was better (or at least statistically insignificant).

5.9.1 *Kruskal-Wallis Test*

Hypothesis testing is required to statistically assert that one algorithm (PSO_AS tuned for quality) is better than another (LK). If the data collected were assumed to be normally distributed data, an F-test or ANOVA test could be performed and the fact that the standard deviation error bars did not overlap would quickly yield the answer that the results were statistically different at the 99% confidence level. Since we do not know the distribution and do not desire to make an assumption, a [Kruskal-Wallis Test] is used. The formula is:

Equation 1 - Kruskal-Wallis Test

$$H = 12 / n(n+1) * (\sum_{j=1}^k R_j^2 / n_j) - 3(n+1)$$

where:

k = number of independent samples (or identical populations)

n_j = number of sample observations from the jth population

R_j = sum of ranks in the sample from the jth population (ranks are assigned by grouping all sample observations)

n = the total number of observations from the k samples

The H value is a chi-squared distribution. In order for the population members to be “distinct” and the results to be statistically relevant at the 99% confidence level, an H value of 6.635 or better must be obtained. In most individual [TSPLIB] problems executed, the ten PSO_AS solutions were all better than the best LK solution. This results in an H value of 14.29. In only one instance (u159) the best LK solution was lower than the best PSO_AS solution. In this case, the H value is 13.17 (see Appendix D.1, Data Sheet #2 for calculation details). Thus, at the 99% confidence level, PSO_AS produces better solutions than LK for all problems tested.

5.10 Experiment 14 – Parallelization of PSO_TSP

Rather than describe the experiment here, a full, unpublished paper is included in Appendix E. In summary, while the paper demonstrates the parallel system produces slightly better solutions for the same CPU time vs. a single computer, the results obtained are less than impressive. Since PSO_TSP did not appear to achieve results competitive with LK, we looked for ways to improve it. PSO_AS is the result.

5.11 Experiment 15 – PSO_TSP vs PSO_AS

Rather than describe the experiment here, a full paper accepted for publication is included in Appendix F. In summary, the native ability of PSO_AS to converge to a solution without heuristic, hill-climbing (such as LK), or Re-hope methods is explored. PSO_AS natively converges to better solutions than PSO_TSP, and does so approximately 30 times faster. This paper justifies the use of PSO_AS.

5.12 Experiment 16 –Tuned PSO_AS vs Published Results of Inver-Over

Table 3- Comparison of PSO-AS with Published Results of Inver-Over

		Inver-Over		PSO_AS	
Instance	Optimum	Result	Time (sec)	Result	Time (sec)
EIL101	629	629.2	7.52	631.4	1.29
LIN105	14379	14379	3.34	14379	0.42
PCB442	50778	51097.5	172.21	51195.3	168.813
PR2392	378032	388095	5366.23	393024.6	19717.9

In all tests, Inver-Over produced better quality of solution than PSO_AS. Inver-Over also produced these solutions in faster time for the two larger problems.

5.13 Experiment 17 –Tuned PSO_AS vs AFIT Research

Table 4 - PSO_AS vs Published Results of GTTS [Hall]

		GTTS		PSO	
Instance	Optimum	Result	Time (sec)	Result	Time (sec)
GR17	2085	2181	7.07	2085	0
GR21	2707	2858	42.64	2707	0
GR24	1272	1338	76.61	1272	0

FRI26	937	941	110.67	937	0
GR48	5046	5710	1549.49	5046	0.03
EIL51	426	522	2340.27	426	0.04
BERLIN52	7542	8649	2768.87	7542	0.02
ST70	675	895	11902.25	675	0.05
EIL76	538	675	17856	538	0.11
BR17	39	39	18.8	87	0
FTV35	1473	1620	388.81	1475	0.01
P43	5620	5652	1005.65	5727	0.02

In all cases, PSO_AS produced solutions in significantly faster time. Those values with a zero represent a time smaller than Windows 2000 can report using the clock() library routine.

PSO_AS also produced the global optimal solution for all symmetric problems. The last three problems are asymmetric, and GTTS produced better solutions for BR17 and P43. While the PSO_AS result for BR17 may seem drastically inferior, the nature of the problem is such that 87 is the second best solution. There are no candidate solutions with fitness values between 87 and 39.

Raw data results for the problems used in [Harder, Kinney, O'Rourke] are included in the Appendix (see D.9). A direct comparison is not possible. These prior AFIT theses test on the TSP with time windows and multiple vehicles. As such, the results from these tests are quite different from using no time windows and only a single vehicle. Results are provided for future reference only.

5.14 Summary of Results and Analysis

Superior solutions are produced with PSO_AS by using a high global percentage (85-95%) and the greedy method. Not using the greedy method produces slightly poorer solutions with improved speed, and with a lower time per generation (hence, better efficiency). Using the greedy method and middle values of global and local trust produce consistent results (both quality of solution and speed) for a wide range of parameter values.

PSO_AS using the tuned parameters produces better solutions than LK for all problems tested at the 99% confidence level.

6 Conclusions and Recommendations

One of the constraints limiting the effectiveness of UCAVs is maximum flight time without refueling (see 2.1). Any improvement in the efficiency of operation increases the effectiveness of the UCAV, since it can obtain more data during the same flight time. By optimizing the planned route, fuel is saved and risk associated with flying over enemy terrain is reduced. Since PSO_AS is shown to produce better solutions than LK, these results directly affect the war-fighting capability of UCAVs!

While these results pertain directly to UCAVs, they also apply directly to numerous unknown works. TSP is the most renown NP-Complete problem, so any improvement in producing solutions affects all other research and applications of TSP.

6.1 Performance of PSO_AS

We can say at the 99% confidence level that the solutions generated by PSO_AS for all problems tested are superior to those generated by LK! Although PSO_AS took more time than LK in the experiments conducted, every solution generated by every single run of PSO_AS was better than all solutions generated by LK. Thus, when LK is given ten times more time, it still doesn't produce solutions better than a single run of PSO_AS. Although PSO_AS took more time than LK for these experiments, if LK is given the same time (or more), it will not produce better solutions.

PSO_AS was also shown to perform significantly better in both speed and quality of solution on symmetric problems against prior AFIT research of GTTS [Hall].

PSO_AS, However, did not outperform published results of Inver-Over. These results are not reproducible even when using code written by the author. It is desirable to reproduce the results to study why it obtains good results and perhaps to incorporate the method into PSO_AS.

By outperforming LK and other published results, PSO_AS exceeds expectations objectives of this thesis.

6.2 Alternatives to PSO_AS

The success of PSO_AS is based on a tour-building method followed by a hill-climbing method (LK). The purpose of the tour-building method is to use known good building blocks to produce likely good candidates that may be on a different hill. Any improvement to LK will also improve PSO_AS. While there may not be a superior hill-climber than LK (it has stood the test of time), it is likely a better tour-building method exists. PSO_AS defines a particular elitist communication pattern (see Appendix F and Appendix B.4) and requires the use of global and local best solutions. This step is similar to the mating selection step in evolutionary computation (see Appendix B.4). Using this tour-build and hill-climb approach coupled with an established evolutionary computation technique may provide even better results.

6.3 Tuning of PSO_AS

One of the consequences of the No Free Lunch Theorem [Wolpert] is that when an algorithm is tuned for performance on a specific problem or class of problems, it will perform less than optimal on other problems. In other words, for every problem there exists a tuning (or tunings) that produces optimal performance, and this tuning may be unique for the problem. While the approach taken to tune PSO_AS was reasonable, and observed generalizations are useful, there is no guarantee these tuned parameters will produce the best performance. Although the “Tuned for Speed” settings did (in general) produce better times than the “Tuned for Quality of Solution” settings, the savings were unimpressive and there were instances where the time taken was longer (and the solution better). It was hoped that similar quality of solutions could be generated in times better than LK. The tuning process used only “smaller” problems, and thus for smaller problems the solution is generated faster than LK. Perhaps a tuning exists for larger problems that will produce good solutions faster, but the tuning process to find those settings will take a long time (ten executions of the 3k city problem took between twelve and fifty hours due to the complexity of the problem and depending on the parameters used).

6.4 Improvements to PSO_AS

While PSO_AS does have performance advantages over LK, there is room for improvement. The version of LK tested and used can be improved. It maps all possible solutions to points that are guaranteed to be opt2 optimal (see A.5). In other words, if LK is run, then opt2 is run, the opt2 will make no changes. Other versions exist that guarantee opt2 and opt3 optimal solutions (thus produce a better quality of solution), but require more time. Also, versions exist that are highly optimized for speed. Since PSO_AS relies on LK, any improvement to LK also improves PSO_AS.

It is also possible that an alternative to LK will yield better results. Reactive Tabu Search has recently gained recognition in optimizing TSP[O'Rourke].

The data structure used for the tour is an array of integers. Although a linked list of integers (from list.h) would seem to be a possible improvement since inversions should only require the change of a few pointers, PSO_AS with this modification took three times longer. Other structures exist for modeling the TSP (graph, heap) and may prove more efficient.

In this implementation, the lowest level object is a Position. In retrospect, an even lower level that should be used is the Tour. Most of the methods for Position operate only on the array containing the Tour. There are also many special cases in the methods to handle the array boundaries. If a Tour contains an array of integers to represent the solution, the zero element of the array should contain the last city, the one element the first city, each subsequent element the city following in the tour, the next to last element the last city (again) and the last element the first city (again). This duplicity of first and last city in the tour will greatly reduce special cases, code size, complexity, and speed of code execution. The cost is that operations that affect these duplicated elements will need to ensure both are updated.

7 References

7.1 TSP

Burkard R.E., V.G. Deineko, R. van Dal, J.A.A. van der Veen and G.J. Woeginger, Well-Solvable Special Cases of the TSP: A Survey, Technische Universität Graz, SFB-Report 52, December 1995.

Clerc, Maurice. Discrete Particle Swarm Optimization Illustrated by the Travelling Salesman Problem. www.mauriceclerc.net. 2000. (Submitted for publication)

Deineko V.G., R. Rudolf, J.A.A. van der Veen, G.J. Woeginger, Three Easy Special Cases of the Euclidean Travelling Salesman Problem, Universität Graz, SFB-Report 17, January 1995.

Demidenko, V.M. and Rudolf R. A Note On Kalmanson Matrices, Gordon and Breach Publishing, 1996.

Garey, M.R. and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, 1979.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan and D. B. Shmoys, *The Traveling Salesman Problem*, Wiley, Chichester, 1985.

Tao, G. and Michalewicz, Z. Inver-over Operator for the TSP, Proceedings of the 5th Parallel Problem Solving from Nature, T. Baeck, A.E. Eiben, M. Schoenauer, and H.-P. Schwefel (Editors), Amsterdam, September 27-30, 1998, Springer-Verlag, Lecture Notes in Computer Science, pp.803-812.

TSPLIB <http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/>

Young C., D. S. Johnson, D. R. Karger, and M. D. Smith, Near-optimal Intraprocedural Branch Alignment, *Proceedings 1997 Symp. on Programming Languages, Design, and Implementation*, 183-193.

7.2 UAV

7.2.1 Predator

http://www.af.mil/lib/afissues/1997/app_b_16.html

http://www.af.mil/lib/afissues/1997/app_b_17.html

http://www.af.mil/news/factsheets/RQ_1_Predator_Unmanned_Aerial.html

<http://www.darpa.mil/tto/programs/ucav.html>

<http://www.dote.osd.mil/reports/FY95/predator.html>

<http://www.fas.org/man/dod-101/sys/ac/ucav.htm>

<http://sun00781.dn.net/irp/agency/usaf/acc/awfc/53w/uavb/>

7.2.2 Global Hawk

<http://www.af.mil/photos/Apr1999/0630pred.html>

<http://www.af.mil/photos/Feb1999/globalhawk.html>

<http://www.defenselink.mil/photos/Feb1997/970220-D-0000G-001.html>

7.2.3 *Dark Star*

<http://www.boeing.com/defense-space/infoelect/darkstaruav/>

http://www.boeing.com/news/releases/1998/news_release_980629a.html

7.2.4 *Air Balloons*

<http://aurora.crest.org/resources/emlists/pvusers/archives/msg00874.html>

7.2.5 *Miniature UCAVs*

<http://www.aerovironment.com/news/news-archive/mav99.html>

7.2.6 *Miscellaneous*

<http://www.unmannedaircraft.com/>

<http://www.vislabs.usyd.edu.au/gallery/aero/>

7.2.7 *Other Research*

Hall, Shane N. "A Group Theoretic Tabu Search Approach to the Traveling Salesman Problem"
AFIT/GOR/ENS/00M-14

Harder, Robert. "A Java Universal Vehicle Router in Support of Routing Unmanned Aerial Vehicles"
AFIT/GOR/ENS/00M-16

Kinney, Gary. "A Hybrid Jump Search and Tabu Search Meta-heuristic for the Unmanned Aerial Vehicle Routing Problem" AFIT/GOA/ENS/00M-5

O'Rourke, Kevin P. "Dynamic Unmanned Aerial Vehicle (UAV) Routing with a Java-Encoded Reactive Tabu Search Metaheuristic" AFIT/GOA/ENS/99M-06

Sezer, Ergin. "Mission Route Planning With Multiple Aircraft & Targets Using Parallel A* Algorithm"
AFIT/GCE/ENG/00M-04

<http://www.au.af.mil/au/aul/bibs/ua/uav.htm>

7.3 Swarm

7.3.1 Representation/Explanation

7.3.2 Application

Carlisle A., Dozier, G. Adapting Particle Swarm Optimization to Dynamic Environments. International Conference on Artificial Intelligence. 2000.

Chapman, Richard and Moore. Application of Particle Swarm to Multiobjective Optimization. IEEE International Conference on Evolutionary Computation. 1999

[see 7.1]Clerc, Maurice. Discrete Particle Swarm Optimization Illustrated by the Travelling Salesman Problem. www.mauriceclerc.net. 2000. (Submitted for publication)

7.3.3 Tuning

Clerc, M. (1999). The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. Congress on Evolutionary Computation, Washington D.C., IEEE

Kennedy, J. and R.C. Eberhart (1995). *Particle Swarm Optimization*. IEEE International Conference on Neural Networks, Perth, Australia, IEEE Service Center, Piscataway, NJ.

Kennedy, J. (2000). Stereotyping: Improving particle swarm performance with cluster analysis. *Proceedings of the 2000 Congress on Evolutionary Computation*, 1507-1512.

Kennedy, J. (1999). Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. *Proceedings of the 1999 Conference on Evolutionary Computation*, 1931-1938.

7.3.4 Ant System

Beckers R., Deneuborg J.L. and Goss S., Trails and U-turns in the Selection of a Path of the Ant Lasuis Niger, In J. theor. Biol. Vol. 159, pp. 397-415, 1992.

Dorigo M., V. Maniezzo and A. Colorni, The Ant System: An Autocatalytic Optimizing Process. Technical Report No. 91-016, Politecnico di Milano, Italy, 1991.

Dorigo M., T. Stutzle ACO Algorithms for the Traveling Salesman Problem. Evolutionary Algorithms in Engineering and Computer Science. John Wiley & Sons, 1999.

7.4 Evolutionary Computation

[BäckBAO00] Bäck, T, Fogel D.B., Micaiewicz, T., Editors, *Evolutionary Computation 1 Basic Algorithms and Operators*, Institute of Physics Pub. 2000

[BäckAAO00] Bäck, T, Fogel D.B., Micaiewicz, T., Editors, *Evolutionary Computation 2 Advanced Algorithms and Operators*, Institute of Physics Pub. 2000

[Bäck96] Bäck, T., *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, New York NY, 1996

- [Baker87] Baker, J., "Reducing bias and inefficiency in the selection algorithm", *Proceedings of the Second International Conference on Genetic Algorithms*, Cambridge MA, 1987
- Banzhaf, Wolfgang and Nordin, Keller, Francone. *Genetic Programming ~ An Introduction* (Chapter 8 – Statistical Tools for GP) Morgan Kaufmann Publishers, Jan. 1998
- Barr, Richard and Golden, Kelly, Resende, Stewart. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1:9-32 1995.
- Corno, Fulvio and Reorda, Squillero, *The Selfish Gene Algorithm: a new Evolutionary Optimization Strategy*. Torino, Italy, 1998.
- Deerman, K., Lamont, G., Pachter, R., "Linkage-Investigating Genetic Algorithms and Their Application to the Protein Structure Prediction Problem", Graduate School of Engineering, Air Force Institute of Technology, Wright Patterson AFB, OH
- Goldberg, D., Kalyanmoy, D., Kargupta, H., Harik., "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms", Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, Urban IL, IlliGAL Report 93004, February 1993
- Goldberg, D., *Genetic Algorithms in Search, Optimization & Machine Learning*, Reading, MA, Addison Wesley, 1989
- Holland J H, *Adaptation in Natural and Artificial Systems*, An Arbor, MI: University of Michigan Press 1975.
- Lee, Chi-Ho, Park, Kim. "Topology and Migration Policy of Fine-Grained Parallel Evolutionary Algorithms for Numerical Optimization", Korea Advanced Institute of Science and Technology, 2000
- Ritter, M., "Clues to mammoth extinction", Associated Press article in the *Dayton Daily News*, Sunday, October 15, 2000, p. 13A
- Runarsson, Thomas, Yao, "Stochastic Ranking for Constrained Evolutionary Optimization" *IEEE Transactions on Evolutionary Computation*, Vol. 4 No. 3 Sept 2000.
- Vazquez, M., Whitley, D., "A Comparison of Genetic Algorithms for the Static Job Shop Scheduling Problem", Computer Science Department, Colorado State University, Fort Collins CO
- White T., and Pagurek B., Towards Multi-Swarm Problem Solving in Networks, *Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS '98)*, July 1998.
- Wolpert D., and Macready, No Free Lunch Theorems for Search. SFI-TR-95-02-010 Feb. 1995.

Kruskal-Wallis Test

A Current Heuristic and Stochastic Approaches To Solution

The TSP has a long history and many heuristic and stochastic methods have been attempted. The most promising heuristic approaches include greedy, insert, re-insert, city swapping, 2-opt, r-opt, and Lin-Kernighan. Stochastic methods that have shown promise include the Ant System and Inver-Over. PSO_TSP is a recent, unproven stochastic method.

A.1 Greedy Heuristic

The greedy heuristic is a straight-forward, simplistic $O(n^3)$ algorithm. Simply choose the next city in the tour that is the closest one available as follows:

- 1) $\forall i \in C$, move i into S and remove i from C . Pick a starting city.
- 2) Choose j such that $j \in C : \text{Min}(c_{ij})$. Find the city that is "closest" to the most recently added city.
- 3) Repeat step 2 until there are no more cities to add.
- 4) Repeat step 1-3 for all starting cities. Each starting city produces a good solution.

The best solution is the resulting solution.

A.2 Insert Heuristic

This is somewhat a greedy, $O(n^2)$ heuristic. Randomly select two cities to start the tour. Randomly select a city not yet in the tour and place it in the tour in such a way to minimize the current partial tour cost. Continue to select and insert cities until all cities are inserted.

A.3 Re-Insert Heuristic

The Re-Insert heuristic is a straight-forward, simplistic $O(n^2)$ algorithm. It takes a current solution and examines each city individually. A search is conducted to determine if there is a better location in the tour for the city. If so, it is moved as follows:

- 1) $\forall i \in C : \text{temp_cost} = c_{(i-1)i} + c_{i(i+1)} - c_{(i-1)(i+1)}$. For all cities in the solution C , calculate the change in cost of removing the city.
- 2) Choose j such that $j \in C : \text{Min}(c_{(j-1)i} + c_{ij} - c_{(j-1)j} - \text{tempcost})$. Find the city where removing i and re-inserting it after j will result in the lowest cost.
- 3) If i is not equal to $j+1$, re-move i and re-insert it after j .
- 4) Repeat steps 1-3 for all cities. The resulting tour is the solution.

This is a hill-climbing, deterministic algorithm. The resulting solution depends on the starting tour.

A.4 City Swapping Heuristic

The City Swapping heuristic is a $O(n^2)$ algorithm. The goal of this algorithm is to select two cities in the tour to swap their positions. These two cities are selected based on being the two cities minimize the tour length as follows:

- 1) $\forall i \in C : \text{temp_cost} = c_{i(i+1)} + c_{(i-1)i}$. For a city in the solution C , calculate the change in cost of “swapping” this city with . . .
- 2) Choose j such that

$$j \in C : \text{Min}(c_{(j-1)i} + c_{i(j+1)} + c_{(i-1)j} + c_{j(i-1)} - c_{(j-1)j} - c_{j(j+1)} - \text{tempcost})$$
. Find the city where removing i and swapping it with j will result in the lowest cost.
- 3) If a j does exist that lowers the cost, swap i and j .
- 4) Repeat steps 1-3 for all cities. The resulting tour is the solution.

This is a hill-climbing, deterministic algorithm. The resulting solution depends on the starting tour.

A.5 2-Opt Heuristic

The 2-Opt heuristic is a $O(f(n)n^2)$ algorithm. The idea behind it is to make two “cuts” in the tour, thus creating two separate segments. One of the two segments is then inverted or reversed and re-joined to the other segment to make a complete tour. Always choose the two cuts that make the greatest improvement and continue to make cuts and improve until no further improvement can be realized and the result is the solution. The $f(n)$ term represents the amount of cuts needed until no further improvement can be realized. While the value of this function is unknown, clearly the larger the problem size, the more cuts are needed (on average).

A.6 r-Opt Heuristic

The 2-Opt heuristic is an r -Opt with a value of two. The r -Opt is an $O(f(n)n^r)$ algorithm. The value of r represents the number of cuts performed with each improvement. In general, the larger the value of r , the more likely it is that the final solution is optimal. Unfortunately, the complexity also increases with r , so values of $r=2$ and $r=3$ are the ones most commonly used.

A.7 Lin-Kernighan Heuristic

The Lin-Kernighan (LK) heuristic is also referred to as a variable r -Opt and is the “supreme” heuristic method found to date. It has complexity of $O(g(n)n^2)$ where $g(n)$ is typically larger than $f(n)$ for the 2-Opt. While it operates in speeds near that for the 2-

Opt, it produces superior solutions. We are conducting an empirical study to examine why LK produces better solutions than 2-Opt. LK is as follows:

- 1) Choose an initial tour.
- 2) Set $G_0^* = 0$. Select any city as a starting city (e.g., city l) and consider one of the edges in the current tour adjacent to this city (e.g., $\{l, k\}$) for removal. Set $p=1$.
- 3) From the other end of this edge (city k) choose an edge that is not in the current tour (e.g., $\{k, i\}$) such that $g_1 = c_{ik} - c_{kl} > 0$. The edge is chosen so that g_1 is maximized.
- 4) Having chosen $\{l, k\}$ to leave and $\{k, i\}$ to enter the solution in the previous iteration, the edge to leave in the p th iteration is uniquely determined. It must be the one of the two edges currently adjacent to city i such that upon removal the set of cities remains connected. In this case, $\{i, j\}$ must leave the solution. Note that adding an edge $\{j, l\}$ reconstructs a tour. Now $G_1^* = g_1 + c_{ij} - c_{jl}$. Let us suppose that $g_1 > G_1^* > 0$. Increment p by 1.
- 5) Edge $\{j, l\}$ is not necessarily the edge chosen to enter the solution at this iteration. That is, we seek to find city q that maximizes $g_p = c_{ij} - c_{jq}$. Suppose $q=m$ and thus the edge chosen to enter is $\{j, m\}$. Calculate $G_p^* = \sum_{s=1}^p g_s$ and then $G_p^* = G_p + c_{mn} - c_{ni}$. Let $G^* = \max\{G_1^*, G_2^*, \dots, G_p^*\}$. We increment p and repeat Step 5 unless:
 - a) no further feasible swaps exist,
 - b) the current configuration is already a tour,
 - c) $G_p \leq 0$, or

$$d) \ G_p \leq G^*.$$

If one of the above conditions holds, construct the tour associated with the best of $\{G_1^*, G_2^*, \dots, G_p^*\}$.

While the above seems quite complex, it is quite easily implemented with a function and some control code as follows:

```
int opti;
tour G;
bool flag;

do{
    flag = false;
    for(int i = 0 ; i < cities; i++)
    {
        opti = i;
        while(opti != -1)
        {
            opti = opt2(G,opti);
            if(opti != -1)
                flag = true;
        }
    }
}while(flag == true)
```

$\text{opt2}(G, \text{opti})$ finds the two cuts that maximizes improvement with one of the cuts being an input (opti), performs the inversion of the segment, and returns the “least optimal” of the two cuts (as per step 5). If no improvement is possible for the given opti cut, it returns -1 . $\text{opt2}(G, \text{opti})$ is $O(n)$.

A.8 Ant System

[Dorigo, Stutzle] describe an algorithm inspired by ants that can find solutions to TSP. In this algorithm, a population of ants creates a tour in a step-by-step process. When an ant is in a given city, it examines the “pheromone” levels leaving its current city and going to cities that the ant has not yet visited. The pheromone levels are used to

determine the probability of the ant traveling to its next city. A higher pheromone level has a higher probability that the ant chooses that path. After all ants have completed a tour, the ant with the best fitness value is allowed to lay down pheromone. Variants typically involve the selection of ants that get to lay pheromone, the types of pheromone (reward and punishment), and the amount or weighting of pheromone used [7.3.4]. Ant Colony Optimization (ACO) is an example of a variant based on pheromone weighting. In ACO, the trail with the highest pheromone level has a large weighting and is highly likely to be picked.

A.9 Inver-Over Genetic Algorithm [Tao]

Inver-Over [Tao] is a genetic operator inspired by the Lin-Kernighan heuristic algorithm. Like Lin-Kernighan, it performs a series of inversions. Unlike Lin-Kernighan, the “second” cut is chosen based on what city follows the current city in the mating parent. Thus the child inherits from the parent this single link. It’s outline is:

```

Random initialization of the population P
while (not satisfied termination-condition) do
{
    for each individual  $S_i \in P$  do
    {
         $S' = S_i$ 
        select (randomly) a city  $c$  from  $S'$ 
        repeat
        {
            if ( $\text{rand}() \leq p$ )
                select the city  $c'$  from the remaining cities in  $S'$ 
            else
            {
                select (randomly an individual from  $P$ 
                assign to  $c'$  the ‘next’ city to the city  $c$  in the selected individual
            }
            if (the next city or the previous city of city  $c$  in  $S'$  is  $c'$ )
                exit from the repeat loop
            inverse the section from the next city of city  $c$  to the city  $c'$  in  $S'$ 
             $c = c'$ 
        }
        if ( $\text{eval}(S') \leq \text{eval}(S_i)$ )
             $S_i = S'$ 
    }
}

```

}
}

A.10 Particle Swarm Optimization and PSO_TSP

A Swarm is a collection of particles. A particle has both a position and a velocity vector. [Kennedy95] gives the “classical” PSO equations where the position and velocity vectors represent physical attributes of the particles. Equation 2 adjusts the particle’s velocity so that the particle moves toward the position of the neighborhood and global best. Equation 3 applies the new velocity to the current position for a single “time period” or generation. Together these equations form simple vector addition.

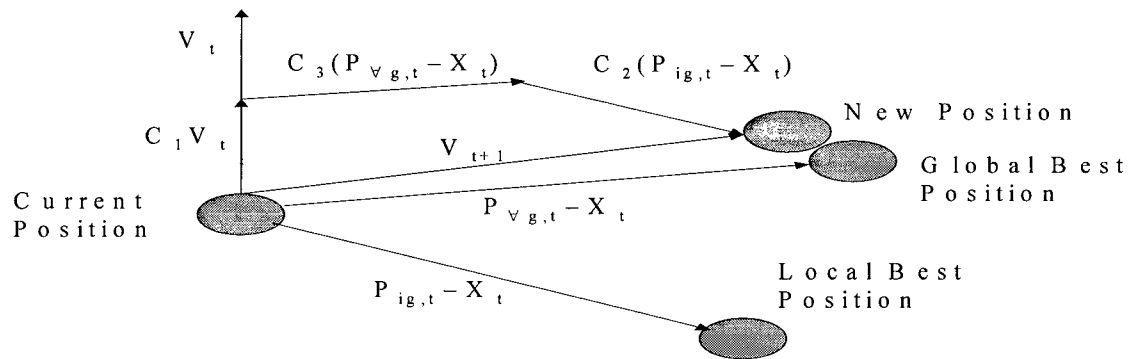


Figure 1 - Illustration of "Classical" PSO

Equation 2 - Calculating a Single Particle's New Velocity

$$V_{t+1} = C_1 V_t \oplus C_2 (P_{ig,t} - X_t) \oplus C_3 (P_{vg,t} - X_t)$$

Equation 3 - "Moving" a Single Particle in a Swarm

$$X_{t+1} = X_t + V_{t+1}$$

The nature of these equations is analogous to planetary bodies orbiting a sun, but is inherently stable. Under these equations (and given appropriate coefficients), all of the particles in the swarm “gravitate” toward the best solution found so far. This produces a local search. It is inherently stable because as the particles get closer to the best solution, there is a *weaker* pull toward it, thus guaranteeing each particle converges (or orbits) near the best solution. This differs from the orbiting analogy since planetary bodies have a stronger pull toward the sun as they get nearer to it, thus are likely to be “flung” away from the sun in a slingshot orbit.

Recent work involves parameter tuning [Kennedy00, Kennedy99], creation of variants [Clerc99], and applying to specific problems [White, Carlisle, Chapman].

A.10.1 PSO Applied to TSP – the PSO_TSP Algorithm

[Clerc] uses the PSO equations, but redefines the meaning of position and velocity to produce good solutions to TSP in the PSO_TSP algorithm. In PSO_TSP, positions represent potential solutions (a list of cities). A Velocity (V) is a list of permutations applied to a particle. Each particle has both a position and an associated velocity. The Velocity Best (VB) is a velocity that maps a particle to the best particle in the swarm found so far. In each generation, each particle is transformed to another particle using a combination of its V and VB. A “No-hope” condition is meant to detect early convergence to a local optimal solution. Possible “No-hope” conditions include 1) all particles in the swarm within a given distance of the best solution, 2) a certain number of generations without discovering a new best solution. When a “No-hope” condition is met, the swarm is re-initialized to random positions to force more exploration. Thus, the Swarm moves and explores the search space according to Equation 2 and 3.

Where:

- 1) V_{t+1} – The particle's new velocity for the next generation
- 2) C_1 – A percentage of the number of "steps" in a velocity to be used. A measure of how much the particle "trusts" its own exploration.
- 3) V_t – The particle's current velocity. A list of permutation steps.
- 4) \oplus – The concatenation of velocity steps.
- 5) C_2 – A percentage of the number of "steps" in a velocity to be used. A measure of how much a particle "trusts" its neighborhood best velocity.
- 6) $P_{i,g,t}$ – The neighborhood (from i to g) best position
- 7) $“-“$ – The difference of two positions is the velocity that will transform the second position into the first position
- 8) X_t – The current position
- 9) C_3 – A percentage of the number of "steps" in a velocity to be used. A measure of how much a particle "trusts" the global velocity.
- 10) $P_{v,g,t}$ – The global best position
- 11) $“+“$ – The transformation of a position using the velocity (yields a position)
- 12) X_{t+1} – The particle's new "moved" position. The position of the next generation.

A.10.2 PSO_TSP Algorithm

Following is an outline of the PSO_TSP algorithm:

1. Initialize the positions and velocities.
2. Determine new particles by applying V_{t+1} to each position (Equation 2,3).
3. Determine the best particle as the one with the shortest tour.
4. If the best solution in a neighborhood is a better solution than VB, save this solution as the new VB.
5. Perform steps 2-4 as above until there is no improvement in the global VB for a set number of iterations (other ending criterion may apply). At this point, we have converged to a local optima and have "No-hope" of finding better solutions. Several methods may be used to find better solutions such as simply starting again from step one without initializing the global best, or using a local search heuristic on the particles and performing steps 2-4.
6. Once all "No-hope" / "Re-hope" methods are exhausted, return the best solution found.

A.10.3 PSO_TSP Example Moving a Particle

Let's look at an example of the way that PSO_TSP moves a single particle:

Let:

$$X_t = \{1,3,5,4,2,1\}$$

$$V_t = \{(3,5),(2,4)\}$$

$$C_1 = C_2 = C_3 = 0.5$$

$$P_{i,t} = \{1,3,4,5,2,1\}$$

$$P_{g,t} = \{3,4,2,1,5,3\}$$

From Equation 2:

$$V_{t+1} = C_1 V_t + C_2 (P_{i,t} - X_t) + C_3 (P_{g,t} - X_t) = 0.5 \{(3,5),(2,4)\} + 0.5 \{(4,5)\} + 0.5 \{(3,1),(4,1),(2,5)\} = \{(3,5)\} + \{(4,5)\} + \{(3,1),(4,1)\} = \{(3,5),(4,5),(3,1),(4,1)\}$$

Note that the list of permutations swaps cities, thus (3,5) means that city 3 is swapped with city 5.

From Equation 3:

$$X_{t+1} = X_t + V_{t+1} = \{3,1,4,5,2,3\}$$

B Evolutionary Computation

This section is a comparison of known evolutionary algorithm (EA) operators with their analogs in the molecular biology realm. See [Bäck96, BäckBAO00, BäckAAO00].

B.1 Comparison of Basic Biological/EA Terminology

Table 5 - Biological and Evolutionary Algorithm Terminology

Biological Terminology	Evolutionary Algorithm Terminology
Nucleotide base	Binary digit, real, integer, programming language element, state machine state...
Codon	Not typically modeled
Gene	Bit string segment, real value vector,
Protein—the basic product of DNA decoding, and the tools that perform the decoding as well as the building blocks that are used in the construction process.	It might be useful to think of this in terms of Evolutionary Programming (EP). In EP, the basic units that are manipulated by the algorithm are programming language and data elements. These elements fit into the process in a manner analogous to how the various bits of protein floating around a cell fit into DNA decoding. The tools used in the construction process are nothing more than functions written in a programming language—perhaps the same language that the EP engine is working on, and finally, the final product of the EP engine is a program made up of the various building blocks. This analogy is rather loose—no EAs that we are aware of actually model the ubiquitous presence and widely varied roles played by proteins in the biological realm.
Allele	Feature value
Locus	String position
Chromosome	Complete bit string
Genotype	Individuals exist at two levels: genotype and phenotype. The genotypic representation of an individual is its genetic material or blueprint for construction. Typically, the various evolutionary operators manipulate individuals at this level. After manipulation this blueprint is used to “decode” an individual into a phenotypic form that is then evaluated for fitness.
Phenotype	
Epistasis	Non-linearity—interactive relationships between genes
Diploid organism—an organism with a set of two chromosomes	It is possible to imagine an EA in which the genetic material is carried in two or more sets of (probably) overlapping chromosomes. It is fairly easy to see how a meiosis operator might then reduce these sets of chromosomes to a single set before individuals are mated.
Haploid organism—an organism with only one chromosome	Most of the EAs in the class references model haploid organisms in that an individual in a population typically has only one chromosome.
Meiosis—the process in which the chromosomes in a haploid organism are reduced into a gamete, which is suitable for recombination	In a loose sense, EAs that use sexual operators model meiosis. It is possible to imagine a diploid EA in which the multiple chromosomes are reduced before using a multiple

with another gamete.	parent operator.	
Mitosis-the process by which a cell splits into two cells. There is no sense of a sexual operator in this process—the chromosomes are copied intact and are identical (or nearly so) in both of the new cells.	Describing an active EA operator analog for mitosis is difficult. There are few compelling reasons to make duplicate copies of an entire EA population element. It is interesting to note that the process of convergence often leads to something like mitosis, in that many members of the population end up as copies of the best population element.	
Population of individuals (tribes, herds, flocks, gaggles, giggles (a group of young girls), etc	Population of elements or individuals in some computer workable form. Unlike their biological counterparts, most EA population elements spend most of their existence being examined and worked on at the genotype level with only a brief existence at the phenotype level for fitness evaluation.	
Mating	Crossover Operators	See discussion below for details
Mutation	Mutation Operators	
Survival	Fitness/Selection Operators	
Evolution	The application of various evolutionary operators through a search process.	
Overall goal: Who knows the meaning or purpose of life? One thing is clear, it once involved the number 42, but not much else is known for certain. There are lots of sub algorithms running all over the place, some of which have readily discernable goals, and these we are trying to understand and model.	Goals, typically some sort of hard optimization problem solution, or search through an enormous solution space.	

B.2 Description of Operators

As alluded to in Table 5 there are four basic elements in every EA: An Algorithm or Strategy in which some Population Elements are manipulated using the various Evolutionary Operators towards satisfying some Overall Goal. This assignment addresses primarily the third element, and the second only in that the representation of the population elements affects which operators are used, and how.

The four main evolutionary operators are fitness, selection, mutation, and recombination (a.k.a. crossover). The development of these operators stemmed from the study of the biological processes involved with evolution. In this section we attempt to play “connect-the-dots” between the EA operators and their biological ancestral concepts. Table 6 shows the relationship between the operators described below, and some instantiations of EAs. Algorithm 1, in Bäck’s unified notation [Bäck96, 63-66], shows how the various components of an EA fit together.

One further thought before we jump into the operators is that they can be divided into classes based upon the number of individuals operated on. The three possibilities are: Asexual, where only one individual is used by the operator, sexual, in which two individuals are worked with, and panmictic, in which more than two individuals are used [Bäck96, 65]. It is interesting to note that asexual and sexual reproduction model popular strategies in the biological world, and are commonly used in EAs, whereas panmictic has no biological foundation, and is rarely used in EAs.

```

t := 0;
initialize P(0) :=  $\{ \vec{a}_1(0), \dots, \vec{a}_\mu(0) \} \in I^\mu$ 
evaluate P(0) :  $\{ \Phi(\vec{a}_1(0)), K, \Phi(\vec{a}_\mu(0)) \}$ 
      where  $\Phi(\vec{a}_k(0)) = \delta(f(\Gamma(\vec{a}_k(0))), P(0))$ ;
while (t(P(t))  $\neq$  true) do
    recombine :  $P'(t) := r_{\Theta_r}(P(t))$ ;
    mutate :  $P''(t) := m_{\Theta_m}(P'(t))$ ;
    evaluate :  $P''(t) := \{ \vec{a}_1''(t), K, \vec{a}_\mu''(t) \}$  :
       $\{ \Phi(\vec{a}_1''(t)), K, \Phi(\vec{a}_\mu''(t)) \}$  where
       $\Phi(\vec{a}_k''(t)) := \delta(f(\Gamma(\vec{a}_k''(t))), P(t - \omega))$ ;
    select P(t+1) :=  $s(P''(t) \cup Q)$ ;
    t := t + 1;
od
```

Figure 33 - Pseudocode for Evolutionary Algorithm

Equation 4 - Evolutionary Algorithm

$$EA = (I, \Phi, \Omega, \Psi, s, l, \mu, \lambda)$$

is called *Evolutionary Algorithm* : \Leftrightarrow

(1) $I = \text{Representational Alphabet}^l$ (Section 3.1)

(2) $\forall \vec{a} \in I : \Phi(\vec{a}) = \delta(f(\Gamma(\vec{a})), \Theta_\delta)$, where $\delta : \mathfrak{R} \times \Theta_\delta \rightarrow \mathfrak{R}^+$ denotes a scaling function and Γ is a decoding function (Section 3.2),

(3) $\Omega = \{m_{\{p_m\}} : I^\mu \rightarrow I^\mu, r_{\{p_c, z\}} : I^\mu \rightarrow I^\mu, r_{\{p_c\}} : I^\mu \rightarrow I^\mu, \dots\}$

where the genetic operators are defined as appropriate for the family of EA (Sections 3.2, 3.3),

(4) $\Psi(P) = s(m_{\{p_m\}}(r_{\{p_c, z\}}(P)))$,

(5) $s : I^\mu \rightarrow I^\mu$, the selection operator, samples individuals according to some sampling strategy (Section 3.2),

(6) the termination criterion, and

(7) $\lambda = \mu$

B.3 Representation

Since computers are binary, the logical choice for representation is a binary string. Pioneers in the field reasoned that representation didn't matter, since regardless of what the representation is its still a binary string in the computer, so why not just keep it a binary string so the computer has an easy time? Yet others showed that representation can make a significant difference in efficiency and effectiveness.

Since it is true that the underlying representation to the computer is a binary string, we must ask why representation makes a difference? To answer this, we must look at how evolutionary computation works, then see the role that representation plays. Evolutionary Computation is a feedback-driven control system. We start with a method of "generating individuals". Once we have the genotypic information of a population, we then look at the phenotypic information. The purpose of the recombination, mutation, and selection operations is to use the information gained from the phenotypic information (i.e., the fitness) to produce "offspring" with even better fitness. An ideal operator would always produce offspring with better fitness values than their parents (given one exists). In such a case, the feedback of the phenotypic information into the genotypic realm always results in the global optimal solution (given enough time). Of course, ideal operators have not been found for most complex problems. In such cases, a good

operator that occasionally (the frequency of which determines the effectiveness of the operator) produces offspring with better fitness values than their parents is used. A good operator is guaranteed to find a “good” solution (usually a local optima), but not to find the global optima (although sometimes it does).

This search for good and ideal operators has lead to the need for various representations. The designer of such an operator must ask why a particular individual received its fitness evaluation? What makes a good individual good, and what makes a bad individual bad? The operator must then use this information to produce offspring. The operator inherently uses domain information. To aid in the performance of the operator, a representation other than binary is often helpful.

An example uses a binary string of 40 bits. The fitness function looks at four bits at a time and adds one to the fitness value only if one of the four bits is a one. Thus, an optimal solution has ten ones and a value of ten. A good operator somehow incorporates “four bits at a time”. For example, it could be crossover that only occurs on the four bit boundaries. While this does give us a good operator and allow us to preserve the binary representation, it is usually better to use a different representation. In this case using an array of 10 integers where each integer may have a value from 0 to 15 would work. In this way, incorporation of the four-bit boundary is inherent in the representation. Choosing an appropriate representation often produces more easily understood, and more efficient code because extra work doesn’t need to be done in code that can be inherent in the representation.

Common representations used are a binary string of ones and zeros (132-5) as discussed previously, a real-valued vector (136-8), an integer string, a permutation (139-49), a finite-state machine (151-4), and a parse tree. Real-valued vectors are used when the problem domain includes real values. For example: find the first 10 real-value constants of the Taylor expansion of $\sin(X) * \exp(X)$. Integer strings are likewise used when they appear in the problem domain (see example above). Permutations are used when a permutation of the optimal solution results in an equivalent solution. Finite-state representations are convenient when the required solutions to a particular problem of interest require the generation of a sequence of symbols having specific meaning.

Another possible representation in these cases is a parse tree, which works well for mathematical (formula) problems and computer programming problems.

B.4 Selection Mechanism: Fitness and Selection

Selection's primary objective is to emphasize better solutions in the search population. This greater emphasis leads to a higher likelihood that the genetic material in a solution is carried into future generations [BackBAO00, 166]. This is directly analogous to the natural process Darwin called Survival of the Fittest. The question of how to rank-order the population arises. In the natural world, this is often the task of predators (or in the case of top predators, lots of very fast and agile prey), but also may fall to great migratory distances, or other difficult to overcome obstacles. In the EA world, this ordering is the responsibility of the fitness function [BackBAO00, 166]. When a solution is decoded, the EAs fitness function examines and records its phenotypic location, and based on that recorded value, ranks the solution.

The three most commonly seen selection operators are: Proportional, Tournament, and Rank-based. All three of these are modeled after the selection operators seen in nature. In proportional selection, as the name implies, population elements are given probabilities for selection directly proportional to their fitness evaluation [BackBAO00, 172, Goldberg89]. According to Grefenstette the process is as follows [BackBAO00173]:

1. Each population elements i is first evaluated to determine its fitness via some fitness function $\Phi(i)$. This may involve a scaling to keep the fitness values inside some defined area., and may also involve some sort of penalty weighting which allows infeasible solutions to compete alongside feasible ones.
2. Once each elements fitness has been determined, a probability distribution,

$$P_{selection}(i) = \frac{\Phi(i)}{\sum_{j=1}^{\mu} \Phi(j)},$$

is created such that the probability of selecting a given population element is proportional to its fitness.

3. The probability distribution is used to select population elements for further evolutionary operators in a number of different ways:
 - a. In steady-state algorithms, one parent at a time is drawn from the distribution
 - b. In generational algorithms, where the entire population is replaced at selection time, the distribution is sampled μ times, in a manner analogous to a roulette wheel.

c. Since the roulette wheel exhibits bias due to the small number of samples, a technique known as Stochastic Universal Sampling was developed that reduces this bias. [Baker87]

3. Depending on the algorithm, other evolutionary operators may now be applied to the selected population members.

This selection operator has received a great deal of attention over the years, indeed the mathematics behind it are very closely linked to the Schema Theorem [Holland]. In the Schema theorem, it is postulated that sub-sequences of genes, known as building blocks, representing hyperplanes in the search space are implicitly manipulated by proportional selection operators. In particular, it is thought that the numbers of those building blocks that contribute to better fitness scores increases exponentially as the search progresses. This selection technique models nature in that certain genes which contribute significantly to survival, such as immunity to a certain disease or an adaptation to specific environmental niche, increases exponentially in future populations.

The next selection operator, Tournament selection, is described by Bickel, as the process of randomly drawing a set of solutions from the population, with or without replacement, and then choosing the element with the highest fitness value as a parent of the next generation. This process is repeated until the proper numbers of parents have been chosen for the next phase of the EA. The primary lure of this technique is its computation cheapness, which derives from the lack of a sorting phase. One disadvantage to this technique is that the mathematics behind the schema theorem don't fit very well into its structure, thus there is often a very high variance in the expected number of offspring that contain certain good building blocks. Fogel discusses several versions of tournament based selection operators: Boltzmann, Soft-Brood, Disruptive, Non-Linear ranking, Competitive Selection, and Variable Lifespan [BackBAO00, 202]. The best natural analog to this EA operator is the tournament based football playoff season. The best team wins each match and is promoted to the next tournament level. This is one of the few cases in which the overall goal of the entire search process is known: It is to determine which team gets to play (and probably beat) the Denver Broncos in the Super Bowl.

Rank-based selection chooses elements from a probability distribution based only on the elements rank-ordering [BackBAO00, 187]. The main difference between proportional selection and rank-based selection is that the fitness of the individual isn't

considered, only its ranking relative to the rest of the population. This helps eliminate problems such as having a single enormously fit individual lead the entire population to premature convergence. In term of natural selection operators, this one closely matches the human habit of (semi)monogamous relationships. In the courting process, a courted individual chooses a mate based largely on the ranking of the chosen individual compared to the other suitors. Since highly attractive individuals (by whatever attractiveness standard is being applied) are ranked highly, they are more likely to be chosen, and may indeed be chosen more than once, but it is pretty rare indeed for a single human to be chosen to mate with basically the entire population of choosers.

When talking about selection operators it is usual to see discussion about the so-called generation gap methods. Sarma and De Jong [Back96, BackBAO00-205] offer up the idea of non-overlapping and overlapping populations. In the non-overlapping population model, the entire parent population is replaced by children (i.e, the parents and their children never compete with each other.) In the overlapping model, parents and children do compete for survival (for some reason this brings to mind the whole Oedipus complex, which is, of course, an entirely different matter.) Traditional notation represents the number of elements in the parent population as μ and the number of children as λ . It is also traditional to denote overlapping populations with $(\mu + \lambda)$ and non-overlapping populations with (μ, λ) . In $(\mu + \lambda)$ system the parents and children compete, and μ members are chosen to continue. In (μ, λ) systems λ is usually $\gg \mu$ and some selection technique is used to pick μ children which then make up the next population. It seems like in the animal (but not always the insect) kingdom, natural selection based on “normal” survival of the fittest is $(\mu + \lambda)$ where both extremes in age decrease the fitness of an individual.

As stated above, the primary objective of the selection operator is to emphasize better solutions. The tool used to distinguish between the solutions, i.e. to determine the value of a particular solution, is the fitness operator. The task of the fitness operator is straightforward—to take the genotype representation of a solution, decode it, evaluate where it is located in the phenotype space with regards to the overall EA goal, and return that evaluation to the selection operator. In practice, however, the problem is anything but simple. In part this complexity stems from the fact that the fitness operator is

typically very problem dependant, it is where the fairly general model of the EA is matched up to the particular problem domain. Very often, determining fitness may involve three operations: decoding, scaling, and evaluation.

In those EAs in which individuals are encoded, i.e. not problem domain object variables, the fitness function must decode them before evaluation [Back96, BackAAO00-4]. Decoding a solution is representation dependent, and the ease with which it can be accomplished plays a large part in the decision as to which representation an EA should use. Scaling is used, among other reasons, to help keep bias in the population down, and to restrict the output from the fitness function to regions which are legal for the selection mechanism [Back96, 111]. The term scaling also seems to crop up in the context of shaping, in which it is used to smooth jagged phenotypic landscapes, or steepen flatter ones. Evaluation is the process of plugging the decoded (if necessary) individual into a function that is expressed in terms of the phenotype space and the optimization goal.

Fitness functions may be single objective, or multiple-objective [BackBAO00, 25]. While single objective functions are more commonly seen in the EA literature, they are actually further from the biological model than their multi-objective functions. One principal that seems to stand out in the molecular biology literature is that most components in natural systems play more than one role, and that the natural fitness function is most definitely multi-objective. In most multi-objective systems, the various factors must be weighted, and the searcher typically returns a set of solution along what is known as a Pareto Front. Finally, fitness functions can be categorized based on what the value that they return is relative to. In fitness functions based on Objective standards, the evaluation of an individual is an objective measure of where the individual falls in the fitness landscape. This is in contrast to Relative and Competitive fitness functions. In the former, a solution's fitness value is relative to some unique solution, or other standard, and in the latter, the individual is rated based somehow on the fitness of the rest of the population [BackBAO00, 12].

B.5 Search Operator: Mutation

In nature, mutation is a change in the genetic material. In this way, new genetic material can be formed. In evolutionary computation its role is either hill climbing or exploration. There are three basic types of mutation [BackBAO00, 237] :

1. Gene mutation: a single gene is affected. This is arguably the most common mutation modeled in GAs. GA mutation is often simply bit-inversion. The probability of mutation is typically low (ranging from .1% to 2%). For each allele, if the probability of mutation is met, then the allele is “flipped”.
2. Chromosome mutation: The gene ordering within the chromosome is changed. Chromosome mutations may often include changes in the number of genes.
3. Genome mutations: Either the number of genomes or the number of chromosomes is changed. Since GAs rarely include multiple chromosomes, this type of mutation is probably rarely modeled.

Since mutation relies on randomness (a very inefficient computational process), it is important to be efficient. A very inefficient method would be to “roll the dice” for every allele. It is far more efficient to “roll the dice” to determine the next allele to be mutated. This applies not only to binary strings, but to all representations.

There are differing mutation operators for various representations. Binary String mutation is discussed above. For real-valued mutation, an allele in the vector of real values is selected for mutation and a random value (usually with mean zero) is added to it. With this type of mutation, the random distribution is significant. A uniform distribution gives tremendous exploration. A Gaussian distribution will not explore nearly as much, and mutated values will be very near the original. A Cauchy distribution is similar to a Gaussian, but has a “fatter tail” so provides for a greater degree of exploration. With permutations, examples of mutations include inverting, swapping, and removing and re-inserting elements. For finite-state machines, there are five mutation modes: 1) change an output symbol, 2) change a state transition, 3) add a new state, 4) delete a state, and 5) change the start state. For parse trees, mutation includes growing the tree by splicing in a randomly generated segment, shrinking the tree by cutting a segment, and Switching the tree by switching two randomly selected nodes in the tree.

B.6 Search Operator: Recombination / Crossover

Recombination in an EA is an analog of the process of information exchange of genetic material that occurs between adjacent chromatids during meiosis. It is through this process that a mixture of genetic information from two parents is combined in a child. This combining of factors from the parent organisms results in exploration of the

fitness landscape by creating novel (from the perspective of the parents) new patterns based on the (presumably) good genes from the parents. This model maps very nicely onto the Schema Theorem [Goldberg89], which attempts to explain how combinations of genes in a chromosome move into future populations. When first exposed to this concept, it seemed to offer a great deal of insight into how EAs work (at least those that resemble genetic algorithms (GA) in the sense that individuals are represented by strings of genes, whether the representation is binary or something more complex.) It is unfortunate that the applicability of the schema theorem has narrowed down to analyzing a single generation, and that it doesn't apply to situations in which the fitness function exhibits a lot of "noise" [BackBAO00, xxxiv]. While crossover is most easily understood in the context of binary strings, it is possible to perform crossover in most EA representations. [BackBAO00, 256-289] provides an excellent discussion of exactly how recombination is performed in various non-binary representations; however, we limit our example discussions to those operations as performed on binary strings. One primary difference between string based and non-string based representations is the care which much be taken to perform crossover in such a way as to produce semantically valid results.

The recombination process usually takes place in three stages: selection of two or more individuals from the population, the determination of one or more crossover locations, and the actual creation of new population members by swapping the material between the selected crossover points [BackBAO00, 257]. The various parameters involved in crossover can be fixed, modified according to a determinist schedule, or dynamically by encoding them into the EA chromosomal structure.

Classification/Comparison of various EA implementations

Genetic Algorithm Name	Representation							Mutation						Recombination						Selection Technique													
For any odd details not captured in this chart, please see the item description below	Binary (132-5)	Real (136-8)	Integer (subset of real)	Permutation(139-50)	Finite State(151-59)	Parse Tree	Other (163-5)	Gene mutation (237-8)	Real (239-43)	Permutations (243-6)	Finite State (246-8)	Parse Tree (248-50)	Other (250-52)						z-point (256-70)	Real (270-4)	Permutations (274-84)	Finite State (284-6)	Parse Tree (286-9)	Other (289-302)	Proportional (172-80)	Tournament (181-5)	Rank-Based (187-93)	Boltzmann (195-203)	Probabilistic?	Other (201-3)			
	X							X												X											X		
	X							X																									
	X							X																									
	X							X																									
									X																								
Simple GA	X							X											X												X		
Stochastic Ranking	X																																
Protein Structure	X							X																									
Fast Messy	X						X																										
Static Job Shop [Vazquez]								X																				X					
Selfish Gene [Corno]								X																							X		
Fine-grained [Lee00]								X																							X		
Inver-over Operator [Tao]			X																									X			X		
Particle Swarm	X																											X			X		
Ant System			X																									X			X		
Genetic Programming																												X			X		

Table 6 - A classification of various Evolutionary Algorithms

B.7 *Selected EC Structure*

While there are a great many possibilities for approaching and solving the TSP, the approach selected is the Swarm. The work of [Dorigo] with AS has shown this approach to be effective. It is logical that since a “socio-evolutionary” method exists in nature that solves the TSP for small dimension problems, the same approach may work computationally for large dimension problems. Unfortunately, there is no guarantee that this approach is any better than any other evolutionary computational method. Specifically, Particle Swarm Optimization was selected. PSO was selected because it is a relatively new approach, thus we are exploring it to see if it offers advantages over other approaches.

B.7.1 Alternatives and Specific EC Representations, Operators

Let’s look at the specific choices made, as well as the alternatives for Representation, Recombination, Mutation, and Selection.

B.7.2 Representation

Because of the problem domain (see B.3), the selected representation is a permutation. While the problem domain compels us to use this representation (other alternatives are inferior per the discussion of B.3), there is still an implementation issue. Is it better to use a linked list or an array of integers? Both will be implemented in order to determine (empirically) the answer.

B.7.3 Recombination

Generically, the recombination method is also a permutation as dictated by the problem domain. Specifically, PSO requires the parents of recombination to be the global best position, the local best position, and the current position. The approach that Inver-Over takes is random selection of parents. The PSO approach should converge faster (and explore less) than more traditional methods.

B.7.4 Mutation

What is the role of mutation? Why do we do it? The standard method of implementing mutation is simply “flipping a bit” (see B.5). As such, it is mostly a hill-climbing technique. Let’s change a bit and see if we improve. A broad view is that it enables more exploration.

The mutation operators available for permutations include inverting, swapping, and removing and re-inserting elements. These operators are all hill-climbing in nature. All

of them are subsets of the Lin-Kernighan (see Lawler) algorithm, which is definitely a hill-climber. In other words, if a position has been Lin-Kernighan optimized, no improvement can be realized by inverting, swapping, or removing and re-inserting elements. Because of this and the fact that the Lin-Kernighan algorithm is incredibly efficient and effective, no explicit mutation operation is used. Instead, the Lin-Kernighan algorithm is used for hill-climbing.

B.7.5 Selection

Selection has two applications. In one sense, we select who is to mate. This is addressed in section B.7.3. When we talk about the neighborhood best or the global best solution, we then have an elitist rank-based selection method.

In the more applicable sense, selection determines which individuals survive to the next generation in the population. In PSO, there is a hybrid method. As far as the personal best solution is concerned, the method is always $(\mu + \lambda)$. The new position competes with the personal best solution. The winner is the personal best solution. As far as the position is concerned, the method is (μ, λ) . The new position always replaces the old position.

B.8 Theory of Execution

With the goal of finding the global optima, we perform a combination of hill-climbing and hill-jumping. The Lin-Kernighan is a most effective hill-climber. We can easily generate many local optima. We then use hill-jumping to combine the local optima and hopefully find a global optima. Often, hill-climbing is needed at this point to ensure the local optimal is truly found.

The Swarm uses this combination of hill-jumping (Recombination) and hill-climbing (Mutation or Lin-Kernighan) followed by fitness evaluation and selection. In so doing, it may very well out-perform a standard Lin-Kernighan algorithm.

C Testing Scripts

These are the “inputs.txt” files used to generate data. See [Design of Experiments]. Copies of these files may also be found on the CD of this thesis.

C.1 Experiment 1 - LK Baseline



C.2 Experiment 2 - PSO_AS Baseline



C.3 Experiment 3 - AS Baseline



C.4 Experiment 4 - Inver-Over Baseline



C.5 Experiment 5 - Course Tuning PSO_AS: 10% Default “Trust”



C.6 Experiment 6 - Course Tuning PSO_AS: 10% Local “Trust”



C.7 Experiment 7 - Greedy PSO_AS: 10% Default “Trust”



greedy10kappatest.txt

C.8 Experiment 8 – Greedy PSO_AS: 10% Local “Trust”



greedytestgamma10.txt

C.9 Experiment 9 – Fine Tuning PSO_AS: 90% Global “Trust”



fine90phi.txt

C.10 Experiment 10 – Fine Tuning PSO_AS: 85,95% Global “Trust”



fine8595phi.txt

C.11 Experiment 11 - Greedy PSO_AS: 85,90,95 Global “Trust”



finetunetestgreedyphi859095.txt

C.12 Experiment 12 – PSO_AS Tuned for Quality of Solution



greedy2890.txt

C.13 Experiment 13 -- PSO_AS Tuned for Speed



2890.txt

C.14 Experiment 17 – Tuned PSO_AS vs Published Results of Inver-Over



Inverfour.txt

//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_lin-kernighan

eil101.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

lin105.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

u159.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

d198.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

tsp225.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

a280.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

pr299.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

linhp318.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

si535.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

pa561.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

u724.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

dsj1000.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

d1291.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

nrw1379.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

fl1577.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

pcb3038.tsp 0 0 0 0 2 1 1 1 0 1 6 100 5 5 7 7 10

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_Swarm_Baseline  
ei1101.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
lin105.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u159.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
tsp225.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
a280.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
pr299.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
linhp318.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
pa561.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
dsj1000.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d1291.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
nrw1379.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
fl1577.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
pcb3038.tsp 80 10 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

PKG3B.txt
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usevl,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_AS-baseline
e11101.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
lin105.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
u159.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
d198.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
tsp225.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
a280.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
pr299.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
linhp318.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
si535.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
pa561.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
u724.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
dsj1000.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
d1291.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
nrw1379.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
fl1577.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10
pcb3038.tsp 100 0 0 0 2 1 1 1 0 0 7 100 4 7 7 7 10

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_AS_with_lin-kernighan  
eil101.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
lin105.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
u159.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
d198.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
tsp225.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
a280.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
pr299.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
linhp318.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
si535.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
pa561.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
u724.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
dsj1000.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
d1291.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
nrw1379.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
fl1577.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10  
pcb3038.tsp 100 0 0 0 2 1 1 1 0 0 5 100 4 5 7 7 10
```

```
PKG3D.TXT
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usevl,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_inver_over
e11101.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
lin105.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
u159.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
d198.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
tsp225.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
a280.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
pr299.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
linhp318.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
si535.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
pa561.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
u724.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
dsj1000.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
d1291.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
nrw1379.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
fl1577.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
pcb3038.tsp 2 0 0 0 2 1 0 1 0 1 6 100 60 7 7 7 10
```

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usevl,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_lin-kernighan  
d198.tsp 10 0 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 0 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 0 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 10 80 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 10 80 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 10 80 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 20 70 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 20 70 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 20 70 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 30 60 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 30 60 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 30 60 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 40 50 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 40 50 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 40 50 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 50 40 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 50 40 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 50 40 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 60 30 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 60 30 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 60 30 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 70 20 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 70 20 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 70 20 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 80 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 80 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 80 10 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
d198.tsp 10 90 0 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
si535.tsp 10 90 0 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

```
u724.tsp 10 90 0 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_lin-kernighan

d198.tsp 0 10 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 0 10 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 0 10 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 10 10 80 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 10 10 80 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 10 10 80 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 20 10 70 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 20 10 70 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 20 10 70 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 30 10 60 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 30 10 60 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 30 10 60 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 40 10 50 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 40 10 50 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 40 10 50 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 50 10 40 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 50 10 40 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 50 10 40 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 60 10 30 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 60 10 30 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 60 10 30 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 70 10 20 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 70 10 20 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 70 10 20 0 2 0 1 1 10 1 5 100 6 5 7 7 10

d198.tsp 90 10 0 0 2 0 1 1 10 1 5 100 6 5 7 7 10

si535.tsp 90 10 0 0 2 0 1 1 10 1 5 100 6 5 7 7 10

u724.tsp 90 10 0 0 2 0 1 1 10 1 5 100 6 5 7 7 10

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_lin-kernighan
d198.tsp 10 0 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 0 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 0 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 10 80 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 10 80 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 10 80 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 20 70 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 20 70 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 20 70 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 30 60 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 30 60 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 30 60 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 40 50 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 40 50 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 40 50 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 50 40 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 50 40 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 50 40 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 60 30 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 60 30 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 60 30 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 70 20 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 70 20 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 70 20 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 80 10 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 80 10 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 80 10 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
d198.tsp 10 90 0 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 10 90 0 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
u724.tsp 10 90 0 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_lin-kernighan  
d198.tsp 0 10 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

```
si535.tsp 0 10 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 0 10 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 10 10 80 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 10 10 80 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 10 10 80 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 20 10 70 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 20 10 70 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 20 10 70 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 30 10 60 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 30 10 60 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 30 10 60 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 40 10 50 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 40 10 50 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 40 10 50 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 50 10 40 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 50 10 40 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 50 10 40 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 60 10 30 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 60 10 30 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 60 10 30 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 70 10 20 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 70 10 20 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 70 10 20 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 90 10 0 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 90 10 0 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 90 10 0 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_lin-kernighan

d198.tsp 1 9 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 1 9 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 1 9 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 3 7 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 3 7 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 3 7 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 4 6 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 4 6 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 4 6 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 5 5 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 5 5 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 5 5 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 6 4 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 6 4 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 6 4 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 7 3 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 7 3 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 7 3 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 8 2 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 8 2 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 8 2 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
d198.tsp 9 1 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
si535.tsp 9 1 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
u724.tsp 9 1 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usevl,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_lin-kernighan  
d198.tsp 0 5 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 0 5 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 0 5 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 1 4 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 1 4 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 1 4 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 2 3 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 2 3 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 2 3 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 3 2 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 3 2 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 3 2 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 4 1 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 4 1 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 4 1 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 5 0 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 5 0 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 5 0 95 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 0 15 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 0 15 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 0 15 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 1 14 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 1 14 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 1 14 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 2 13 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 2 13 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 2 13 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 3 12 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 3 12 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 3 12 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 4 11 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 4 11 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 4 11 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 5 10 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 5 10 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 5 10 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 6 9 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 6 9 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 6 9 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

d198.tsp 7 8 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 7 8 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 7 8 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 8 7 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 8 7 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 8 7 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 9 6 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 9 6 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 9 6 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 10 5 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 10 5 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 10 5 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 11 4 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 11 4 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 11 4 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 12 3 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 12 3 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 12 3 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 13 2 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 13 2 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 13 2 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 14 1 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 14 1 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 14 1 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 d198.tsp 15 0 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 si535.tsp 15 0 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10
 u724.tsp 15 0 85 0 2 0 1 1 10 1 5 100 6 5 7 7 10

//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,rehope3,runs;test_for_lin-kernighan

d198.tsp 1 9 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 1 9 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 1 9 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 3 7 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 3 7 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 3 7 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 4 6 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 4 6 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 4 6 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 5 5 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 5 5 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 5 5 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 6 4 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 6 4 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 6 4 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 7 3 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 7 3 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 7 3 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 8 2 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 8 2 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 8 2 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 9 1 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 9 1 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 9 1 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 0 5 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 0 5 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 0 5 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 1 4 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 1 4 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 1 4 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 2 3 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 2 3 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 2 3 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
d198.tsp 3 2 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
si535.tsp 3 2 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
u724.tsp 3 2 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10

d198.tsp 4 1 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 4 1 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 4 1 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 5 0 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 5 0 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 5 0 95 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 0 15 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 0 15 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 0 15 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 1 14 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 1 14 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 1 14 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 2 13 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 2 13 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 2 13 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 3 12 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 3 12 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 3 12 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 4 11 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 4 11 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 4 11 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 5 10 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 5 10 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 5 10 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 6 9 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 6 9 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 6 9 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 7 8 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 7 8 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 7 8 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 8 7 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 8 7 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 8 7 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 9 6 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 9 6 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 9 6 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 10 5 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 10 5 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 10 5 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10

d198.tsp 11 4 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 11 4 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 11 4 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 12 3 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 12 3 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 12 3 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 13 2 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 13 2 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 13 2 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 14 1 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 14 1 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 14 1 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 d198.tsp 15 0 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 si535.tsp 15 0 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10
 u724.tsp 15 0 85 0 2 0 1 1 10 1 5 100 2 5 7 7 10

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_lin-kernighan  
eil101.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
lin105.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u159.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d198.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
tsp225.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
a280.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
pr299.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
linhp318.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
si535.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
pa561.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
u724.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
dsj1000.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
d1291.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
nrw1379.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
fl1577.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10  
pcb3038.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10
```

PKG46.txt

```
//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usevl,preinit,sizex,movetype,moveheu,rehope2,r  
ehope3,runs;test_for_lin-kernighan  
eil101.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
lin105.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u159.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d198.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
tsp225.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
a280.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
pr299.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
linhp318.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
si535.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
pa561.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
u724.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
dsj1000.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
d1291.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
nrw1379.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
fl1577.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10  
pcb3038.tsp 2 8 90 0 2 0 1 1 10 1 5 100 6 5 7 7 10
```

PKG47.TXT

//f_string,kappa,gamma,phi,trace,queens,procs,vels,ins,nei,usev1,preinit,sizex,movetype,moveheu,rehope2,r
ehope3,runs;test_for_lin-kernighan
eil101.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10

lin105.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10

pcb442.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10

pr2392.tsp 2 8 90 0 2 0 1 1 10 1 5 100 2 5 7 7 10

D Excel Spreadsheets of Raw Data and Charts

These are the Excel spreadsheets or raw data created from the experiments. The associated files are also on the CD of the thesis. In the “one” row above the spreadsheet data is the name of the “inputs.txt” file used to create the data.

D.1 Experiment 1 - LK Baseline, Experiment 2 – PSO_AS Baseline, Experiment 12 – PSO_AS Tuned for Quality of Solution, Experiment 13 – PSO_AS Tuned for Speed



D.2 Experiment 3 – AS Baseline



D.3 Experiment 4 – Inver-Over Baseline



D.4 Experiment 5 – Course Tuning PSO_AS: 10% Default “Trust”, Experiment 6 – Course Tuning PSO_AS: 10% Local “Trust”,



D.5 Experiment 7 – Greedy PSO_AS: 10% Default “Trust”,



D.6 Experiment 8 – Greedy PSO_AS: 10% Local “Trust”



D.7 Experiment 9 – Course Tuning PSO_AS: 90% Global “Trust”, Experiment 10 – Fine Tuning PSO_AS: 85,95% Global “Trust”



D.8 Experiment 11 - Greedy PSO_AS: 85,90,95 Global “Trust”



D.9 Experiment 18 – Tuned PSO_AS vs AFIT Published Results

E Heterogeneous Implementation of a Particle Swarm Optimization

The file “parallelPSO_TSP.doc” is an unpublished paper included on the CD. The following link should automatically open the paper:



parallelPSO_TSP.doc

Heterogeneous Implementation of a Parallel Particle Swarm Optimization Using Windows 2000/Linux Redhat 6.2 On a Myrnet 100 MB Network

Barry R. Secrest and Steven Michaud
Air Force Research Laboratory
Wright-Patterson AFB OH

Dr. Hartrum
Air Force Research Laboratory
Wright-Patterson AFB OH

{barry.secrest, steven.michaud}@afit.af.mil

Abstract: *Particle Swarm Optimization (PSO) has been shown to provide good answers to the Traveling Salesman Problem. While not as powerful as some specific algorithms, its adaptability makes it a good candidate for a wide range of combinatorial problems. This research demonstrates a Java implementation and analyzes the performance of it in a Windows 2000/Linux Redhat 6.2 Heterogeneous network.*

Keywords: Pile of PCs, Particle Swarm Optimization, Traveling Salesman Problem, Linux Redhat 6.2, Windows 2000, Parallel Algorithms, Evolutionary Computation

1 Introduction & Background Information

Each and every master's student at AFIT is required to conduct research and work on an "Air Force" related problem to satisfy his or her "thesis" requirement. Like most students, many projects are undertaken with this goal in mind. The selection of this problem is intended to "kill two birds with one stone." Obviously the first reason is to fulfill the Distributed Operating Systems (CSCE 689) coursework requirements while the second is merely to provide additional insight and understanding of particle swarms and

Submitted in partial fulfillment of the requirements for the Master degree at the Air Force Institute of Technology, Wright-Patterson AFB OH.

how they might be used to help solve mission planning of a set of Blackhawks.

1.1 Air-Force Related Problem

The Blackhawk is an Unmanned Combat Air Vehicle (UCAV). Since it is unmanned, it can be built smaller and designed to take gravitational forces in a way that a human simply couldn't stand. Its mission is reconnaissance. It flies at a constant height above enemy territory and gathers data using various antennae.

Mission planning for the Blackhawk is computationally challenging. In what way do you fly the Blackhawk over the territory to obtain the data? It is desirable to travel the shortest path, since that path will allow us to obtain the data swiftly, minimize flight time thus minimizing the possibility of being shot down by enemy fire, as well as consuming less fuel. Given a set of coordinates (either from earlier satellite or Blackhawk mission reconnaissance), mission planning is then easily mapped to the ever-popular Traveling Salesman Problem (TSP) – a well-known NP-complete problem.

PSO is inspired by the way swarms of insects, though individually not very intelligent, can produce nearly optimal solutions to highly complex problems. Experiments have been shown where ants (yes, the real insects) have been placed in an environment with an anthill and nearby food. The paths they followed to the

food were nearly optimal. In fact, think of the food and anthill as cities and suddenly the ants are finding solutions to TSP.

1.2 AFIT Parallel Lab

The AFIT laboratory consists of a Pile of twenty-two computers, ranging from 333MHz to 600MHz. Each system, with the exception of the two servers are dual (Linux Redhat 6.2/Windows 2000) bootable. The communications backbone is capable of 100Mbps via two Myrnet 100 MBps Fast Ethernet switches (that are interconnected via one-gigabit switch).

1.3 Source Code

We obtained a copy of a C++ serialized version of the PSO software tailored to solve a TSP problem and modified it for a previous class to utilize the MPIPro communications software libraries in a homogenous operating system environment. While we could have modified the code to make it run in a heterogeneous environment, the effort would have required us to install new software and implement a crude form of inter-processor communications via file I/O. A previous AFIT student, Captain Chris Bohn, implemented a heterogeneous parallel program by using MPI software and an agreed upon file with a predetermined data format. The file was both written to and read by each process. His MPI heterogeneous effort was accomplished in the AFIT parallel lab where each system is dual boot into either Linux or Windows 2000.

We thought seriously about taking this route which, on one hand would have made the coding considerably easier—our code was already programmed using MPI and C++, we'd have just had to add file IO and a few other MPI calls to identify cross-platform machines that were going to run the PSO software—but we decided/opted not to do that to learn more about Java and to avoid some of the difficulties that Captain Bohn had when he implemented cross-platform parallel, MPI-based software. As a result, we morphed the C++ code into Java to take advantage of the fact that it runs cross-platform, has built-in communications capability, and isn't too dissimilar to C++.

The program is designed in the following manner: each computer executed the serial software virtually unchanged. When a new best solution (for that computer) was found, it sent this to its neighbor. When a new best solution is received from a neighbor, information from the solution is used to further direct the search. The efficiency, effectiveness, speedup, scalability, and quality of solutions of this software and are discussed later.

1.4 Communications

1.4.1 Packages

Java is a true object oriented language unlike C/C++, which is hybrid and has built-in support for multithreading and synchronization of multiple threads.

MPIPro consists of a set of library functions that can be linked to ones' program providing a means of communicating between computers so they may execute parallel programs.

1.4.2 PSO Communication Methodology

The original version of the PSO software was serial in nature and required no communications capability.

The MPI modified version, the work done in a previous class, used MPI library routines to simulate a logical ring network—starting node is 0, and passes info to node 1, node 1 passes info to node 2, etc. with the last node passing information back to node 0, the starting node.

Our initial intent was to imitate the MPI version of the ring; however, we found out the hard way that it is very difficult for Java to set up both client and server on each processor—while it can be done, it is not a simple matter and thus we decided upon a client-server configuration that simulates a logical ring by having the server receive info from a given node and passing it on to the next “logically higher numbered” node, with the last passing to the first in the ring.

The PSO server is started and waits for a pre-specified number of clients to register. Once all clients have registered the processors are allowed

to start working on the TSP problem at hand. The server listens on each respective port and passes along any new "best" solution found. As clients register and deregister with the server this process is continued until a predetermined number of generations have occurred or some other criteria is met. Upon completion, all clients once again report in to provide the server their best solution and the server reports the final results.

2 Software Methodology

2.1 PSO Algorithm

2.1.1 Serialized PSO Algorithm

PSO-TSP uses particles (referred to as the Particle Swarm) to represent potential solutions. A Velocity (V) is a step-by-step mapping of a particle to another particle. Each particle has an associated velocity. The Velocity Best (VB) is a velocity that maps a particle to the best particle in the swarm found so far. In each generation, each particle is transformed to another particle using a combination of its V and VB. A "no-hope" condition causes a re-initialization of the swarm to prevent early convergence to a local optimal solution. The Swarm moves and explores the search space according to the following equations:

$$V_{t+1} = C_1 V_t \oplus C_2 (P_{ig,t} - X_t)$$

Equation 1 – Calculating a Single Particle's New Velocity

$$X_{t+1} = X_t + V_t$$

Equation 2 – "Moving" a Single Particle in a Swarm

Where:

- 1) V_{t+1} – The particle's new velocity for the next generation
- 2) C_1 – A percentage of the number of "steps" in a velocity to be used. A measure of how much the particle "trusts" its own exploration.
- 3) V_t – The particle's current velocity
- 4) \oplus – The concatenation of velocity steps
- 5) C_2 – A percentage of the number of "steps" in a velocity to be used. A measure of how much a particle "trusts" its neighborhood best velocity (VB).
- 6) $P_{ig,t}$ – The neighborhood (from i to g) best particle
- 7) "-" – The difference of two particles is the velocity that will transform the second velocity into the first velocity
- 8) X_t – The current particle
- 9) "+" – The transformation of a particle using the velocity (yields a particle)
- 10) X_{t+1} – The "moved" particle. The particle of the next generation.

2.1.2 Parallel PSO Algorithm

The following define the steps taken by the parallel PSO algorithm.

1. Randomly initialize the particles and velocities on all processors.
2. Determine new particles by applying V to each particle.
3. Determine the best particle (including the prior best particle, as well as any particles passed by other processors).
4. If VB for a given processor is a better solution than the prior VB, pass this particle to neighboring processor via the server.
5. Determine a new V for each particle using the old V and VB.
6. If the number of desired iterations is reached, return the best particle. Else if pre-convergence occurs (a "no-hope" condition representing having found a local optima), go to step 1. Else go to step 2.

2.2 Lab Configuration

While the AFIT laboratory has 19 systems with 20 processors (1 dual processor machine), we intended on using only use the eight 600MHz processors; however, one of the systems was inoperable and as a result our tests we done on 2, 4, and 6 machines only. Note: Our initial decision to only use 8 machines was based on the fact that the previous version of the PSO software was only tested on eight machines. By keeping this number the same we hope to be able to draw viable conclusions about our findings.

2.3 Mapping the Algorithm onto the Hardware

Other logical configurations than the ring could have been used; however by only communicating when a client has found a new best the amount of communications can be kept to a minimal—remember, the server will only pass along the global best when a client communicates it's latest best. It has been shown that higher connectivity between islands results in fewer iterations per processor for convergence. While faster convergence may seem desirable, it may result in poorer solutions due to pre-convergence of a local optima. Also, real performance of parallel algorithms depends on maximizing processing and minimizing communications. The star configuration was selected as a result of limitations of the Java programming language and the fact that it doesn't do non-blocking communications well.

3 Approach

TSPLIB is a library of known TSP problems used as a benchmark. While the previous version (the MPI version) was run against four of the TSPLIB problems, we opted to only test out the Java version on a single TSPLIB problem—the idea for this project is to demonstrate the implementation in an O/S heterogeneous environment. The test was run five times on two, four, and six processors. Metrics looked at were time to completion vs. # of processors, overall effectiveness of the algorithm vs. # of processors, and time to converge vs. # of processors.

4.1 Output Screens

[illegible]

Figure 1 - Server Output

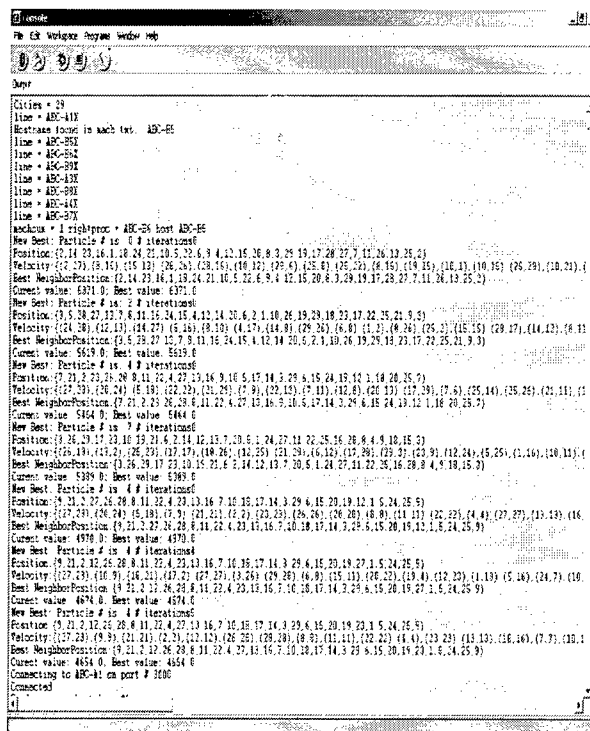


Figure 2 - Client Output

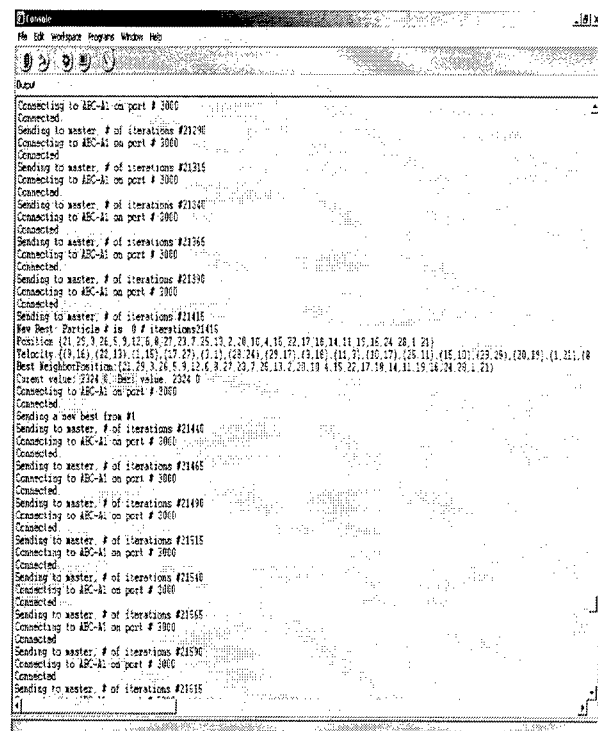


Figure 4 - Client Output - Receiving Solution

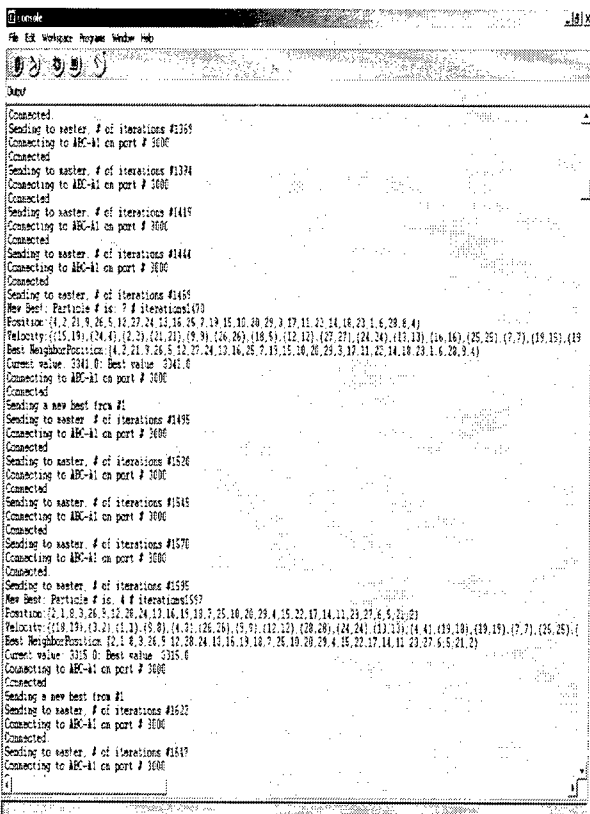


Figure 3 - Client Output - Passing Solutions

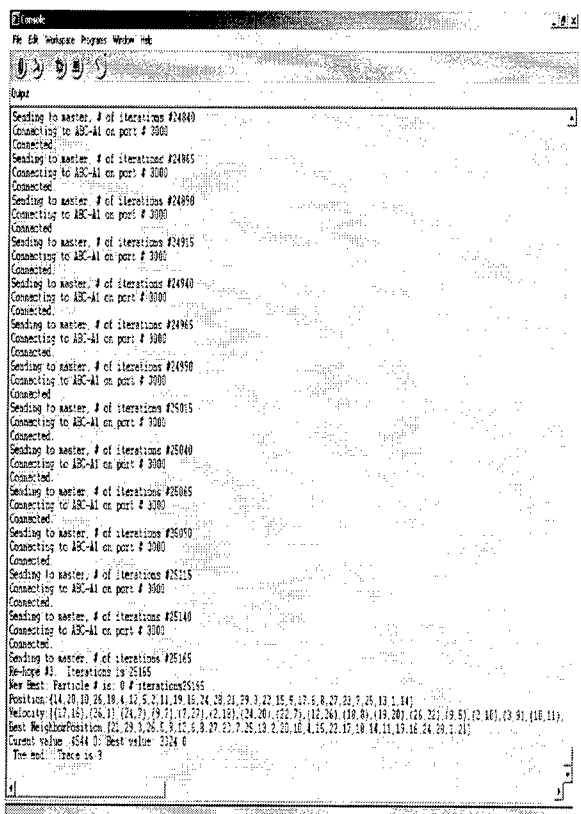


Figure 5 - Client Output - Completion

4.2 Results

File	MPI PSO Best	Java PSO Best 2/4/6 Processors	Optimal
Br17	86	---	39
Bays29	2142	2379/2260/2033	2020
Ftv33	1687	---	1286
P43	5752	---	5620

**Table 1: Comparison of Solutions.
(Only Bays29 Tested)**

It appears that the more processing power used to solve the Bays29 TSP benchmark, the better the solution found. Of course this makes entire sense because we are able to explore and exploit far more with multiple processors than with a single one. It would be interesting to see just how well the algorithm performs against the other TSP benchmark problems, as well as, the performance of the swarm once other enhancements are incorporated into the algorithm. Should the algorithm prove fruitful, the next step would be to run it up against larger TSP benchmarks.

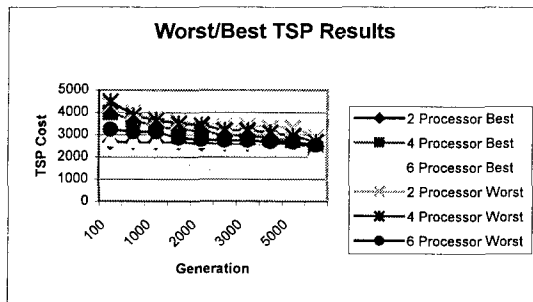


Figure 6 – Worst/Best TSP Results

This figure demonstrates that from the get-go results obtained from generation to generation tend to be better as additional processors are used to solve the problem. Again common sense tells us that the more work done to solve the problem, the better chances of solving it. From a Genetic Algorithm point of view, the more processors used to solve the problem the more landscape is searched in finding the solution.

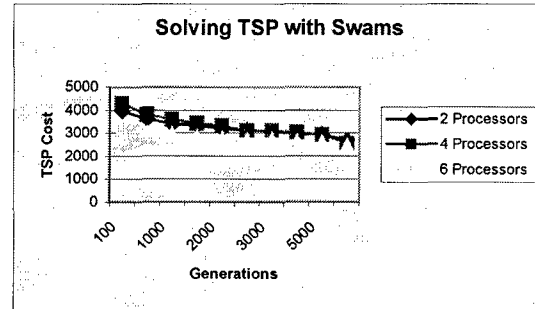


Figure 7 – Solving TSP with Swams

This figure compares the performance from each of the 2, 4, and 6 processor runs. From the start, 6 processors clearly is more effective in solving the problem. On the other hand, it is clear that as the number of processors are increased, the average number of generations used to solve the problem increases, which in turn, results in an overall increase in computational time (see Table 3).

Number of Processors	Average TSP Cost	Average # of Generations
2	2552.5	10956.7
4	2442.3	12254.7
6	2266.5	14665.8

Table 3 – Average TSP Cost & Generations

4.3 Efficiency

Efficiency is the measure of how much work (time spent doing calculations) is being performed compared to the overhead of sending messages. By design, this algorithm is efficient. The message size is small, and messages are only passed when a new optima is produced. It was observed that all processes sent several messages (up to four) within the first second. After that, messages came about once per processor per five seconds. With so little message passing, efficiency is high.

4.4 Scalability

In this context, scalability has two aspects. 1) Can we use even more processors, and if we do, will the solution generated be "better" (either faster or more precise). 2) Can we throw even larger problems at it? To answer the first question, yes, it is scalable. The Island approach allows you to use as many processors as are available without seriously affecting efficiency. While the quality of solutions does improve with

more processors, the time to produce solutions will generally increase as more processors tend to find better solutions for a longer period of time and as a result keeps the algorithm going—as the termination is based on not obtaining a “new” best within a specified number of generations.

The second question is trickier. In theory, we should be able to use any size problem. Attempting to use problem sizes of greater than fifty cities resulted in stack over-flow errors for the original serial version. While it wasn't tested, the Java version should not have this limitation and should be able to handle problem sizes near 32,000 cities.

5 Conclusions

5.1 Java Implementation Performance on Different Platforms

In an attempt to provide additional insight into the overall operational speed of Java on different platforms, we pulled up a recent article in the August 2000 issue of Linux Journal entitled “Comparing Java Implementations for Linux”. The author discusses his “home made” benchmark and readily admits that it doesn't cover all aspects of the language but in does do what most object oriented programs do and that is create objects and call methods from that object. The benchmark creates 500,000 objects and calls a method of that object. The results are the elapsed time—the total time to create and call a method half a million times.

The table below is an abbreviated version found in the article and merely shows elapsed time between running the benchmark application in Java code via a compiler or via a just-in-time (JIT) interpreter/compiler and finally in a C++ natively compiled code. It must be noted that the benchmark was run on a homogenous operating system platform and on a single 233MHz CPU with 128 Mbytes of RAM under Red Hat Linux 6.1.

Note: The benchmark does not address communications but instead compares the performance of the native compiler, the

interpreter (Intptr), and the just-in-time (JIT) Java interpreter/compiler.

Java Implementation	Elapsed Time	Mode
Sun JDK 1.2.2	4815	Intptr
Sun JDK 1.2.2	4973	JIT
Blackdown 1.2.2	4869	Intptr
Blackdown 1.2.2	4123	JIT
Blackdown 1.1.6	7356	Intptr
Blackdown 1.1.6	6207	JIT
Cygnus Codefusion	2854	Native

Table 4 - Java Benchmark Results

It is clear from the preceding table that native code is far superior to Java, but is also clear that not is Java is equal—at least with respect to execution time.

In a nutshell this means that running in a heterogeneous environment “should” have a negative affect of running different flavors of Java that have different performance specs.

We demonstrated that a means of using a Particle Swarm in a heterogeneous parallel environment is reasonably successful. Most of the problems the algorithm could handle were too trivial to present a challenge. With two processors, a solution was found usually under a second. For the most difficult problem the algorithm could handle, we demonstrated that more processors did slightly improve the quality of solutions and/or the time needed to find the solution. In all cases, the Swarm proved it could find good solutions to the TSP, but it doesn't perform as well as other more specialized algorithms. To remedy this, further study needs to be done to determine the best means in which to pass “good” information about what makes a good solution good.

References

Bailey, Glenn and others, Unmanned Aerial Vehicle (UAV) Route Selection Using Reactive Tabu Search. Military Operations Research, 1999.

Bohn, Chris, CSCE 689 Distributed Operating Systems, AFIT School of Engineering, 1999.

Bondy, J.A. and Murty U.S.R., Graph Theory with Applications, North-Holland, 1974.

Brualdi, Richard A. Introduction Combinatorics. North-Holland, 1977.

Chapman, Richard and Moore. Application of Particle Swarm to Multiobjective Optimization. IEEE International Conference on Evolutionary Computation. 1999

Christofides. Graph Theory, An Algorithmic Approach. Academic Press, 1975.

Clerc, Maurice. The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. IEEE International Conference on Evolutionary Computation. 1999

Clerc, Maurice. Discrete Particle Swarm Optimization Illustrated by the Travelling Salesman Problem. www.mauriceclerc.net. 2000.

Gould, Ronald. Graph Theory. The Benjamin/Cummings Publishing Company, Inc., 1988.

Hirsch, Michael, "Comparing Java Implementations for Linux", LINUX Journal, August 2000, SSC Publications.

Kumar, Vipin and others. Introduction to Parallel Computing. The Benjamin/Cumming Publishing Company, Inc. 1994.

Skiena, Steven S. The Algorithm Design Manual, Springer-Verlag, New York, 1998.

West, Douglas B., Introduction to Graph Theory, Prentice Hall, 1996.

F Communication in Particle Swarm Optimization

The file “PSO_Ants v2.doc” is a paper accepted for publication (but not yet edited and sized down) included on the CD. The following link should automatically open the paper:



"PSO_Ants v2.doc"

Communication in Particle Swarm Optimization

Illustrated by the Traveling Salesman Problem

Barry R. Secrest
Air Force Institute of Technology
Wright-Patterson AFB OH

Gary B. Lamont
Air Force Institute of Technology
Wright-Patterson AFB OH

{barry.secrest, gary.lamont@afit.af.edu} Fax#: (937)656-4055

Abstract *It is easily believed that the equations for Particle Swarm Optimization (PSO) define a specific algorithm. In fact, the equations alone provide merely the communication pattern, with some of the algorithm details determined by the equation sub-definitions. It has been demonstrated that Particle Swarm Optimization provides good answers to many optimization problems including the Traveling Salesman Problem (TSP). We present a revised PSO algorithm inspired by the Ant System (AS), thus demonstrating that the original PSO equations themselves do not define the algorithm.*

This paper analyzes the differences between these PSO algorithms. The nature of Swarms is revealed, allowing others to more easily construct generic Swarm algorithms.

Keywords: Particle Swarm Optimization, Traveling Salesman Problem, Evolutionary Computation, Ant System

1 Introduction

Particle Swarm Optimization is inspired by swarms of insects [Kennedy]. Though individually not very complex, they can produce nearly optimal solutions to highly complex problems. Experiments have been presented where ants have been placed in an environment with an anthill and nearby food [Beckers]. The paths they followed to the food were nearly optimal. This led to using ant foraging behavior as a computational problem-solving method [Dorigo].

Section two of this paper discusses the TSP, a Particle Swarm Optimization algorithm (PSO_TSP) presented by [Clerc], and an algorithm known as the Ant System (AS) as presented by [Dorigo] and used to solve the TSP. Section three presents a new algorithm (PSO_AS) in detail that synthesizes ideas from both PSO_TSP and AS. An example is given to illustrate how PSO_AS works. In section four, an experiment is conducted to empirically illustrate how PSO_AS converges compared to PSO_TSP. Why this algorithm works is explored and discussed in section five to aid others in constructing generic Swarm algorithms.

Submitted in partial fulfillment of the requirements for the Master of Electrical Engineering degree at the Air Force Institute of Technology, Wright-Patterson AFB OH.

Conclusions and future work are discussed in sections six and seven respectively.

2 Background

2.1 Travelling Salesman Problem

The Travelling Salesman Problem (TSP) is defined as follows. Given an $n \times n$ distance matrix $C = (c_{ij})$, find a permutation, $\pi \in S_n$ that minimizes the sum $\sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} + c_{\pi(n)\pi(1)}$ [Lawler]. The Salesman must visit all cities from 1 to n exactly once in such a way to minimize the distance traveled. It has direct, real-world application to routing problems in general (mail or other kinds of delivery), and as a NP complete problem has indirect relevance to many other real-world problems. It is undoubtedly the most famous NP complete problem.

The problem is often classified by characteristics of the distance matrix. In symmetrical TSPs, $c_{ij} = c_{ji}$, thus all values in the matrix are symmetrical through the diagonal. If it is not symmetric, it is asymmetric. In triangular TSPs (Δ TSP) $\forall i, j, k \in C : c_{ik} \leq c_{ij} + c_{jk}$, thus given the distances between any three cities, a physical triangle can be formed to represent those three cities' distances. In Euclidean TSPs,

$$\forall i, j \in C : c_{ij} = \sqrt{(a_i - a_j)^2 + (b_i - b_j)^2},$$

thus the cities can be mapped to a physical grid. Euclidean TSPs are both symmetric and triangular, but these characteristics alone do not imply

Euclidean. It is also possible for a matrix to be triangular and asymmetric. Other matrix classifications include Kalmanson, Demidenko, or Supnic conditions, which are trivially solved by looking at the plot of the points.

Problem complexity is often related to matrix classification. For Kalmanson, Demidenko, and Supnic matrices, optimal solutions can be generated in polynomial time. Δ TSPs are easier because the triangular relationship is used heuristically to produce solutions. Symmetric TSPs are easier than asymmetric because a multi-city path can be reversed (inverted) without having to recalculate the path cost. Also, of all possible tours, there are twice as many solutions (the forward tour and the backward tour), thus only half of the search space needs to be searched to find the optimal solution. Euclidean TSPs are considerably easier to find good solutions for, yet no polynomial algorithm has been discovered to find the optimal solution.

The Lin-Kernighan algorithm is the most efficient heuristic method for finding good solutions to the TSP known to date, but doesn't work as well with asymmetric TSPs. Perhaps the most efficient stochastic method is the inver-over operator [Tao], which may produce better solutions than Lin-Kernighan, but requires more time. It is limited to working well only for symmetric, Δ TSP. For further information, see the book by [Lawler].

2.2 Particle Swarm Optimization

A Swarm is a collection of particles. A particle has both a position and a velocity vector. [Kennedy95] gives the

“classical” PSO equations (see Equations 1 and 2) where the position and velocity vectors represent physical attributes of the particles. Equation 1 adjusts the particle’s velocity so that the particle will move toward the position of the neighborhood and global best. Equation 2 applies the new velocity to the current position for a single “time period” or generation. Together these equations form simple vector addition.

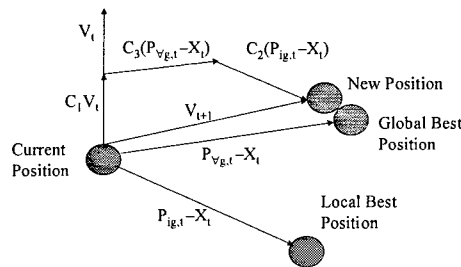


Figure 1 – Illustration of “Classical” PSO

The nature of these equations is analogous to planetary bodies orbiting a sun, but is inherently stable. Under these equations (and given appropriate coefficients), all of the particles in the swarm “gravitate” toward the best solution found so far. This produces a local search. It is inherently stable because as the particles get closer to the best solution, there is a *weaker* pull toward it, thus guaranteeing each particle converges (or orbits) near the best solution. This differs from the orbiting analogy since planetary bodies have a stronger pull toward the sun as they get nearer to it, thus are likely to be “flung” away from the sun in a slingshot orbit.

Recent work involves parameter tuning [Kennedy00, Kennedy99], creation of

variants [Clerc99], and applying to specific problems [White, Carlisle, Chapman].

[Clerc] uses the same equations, but redefines the meaning of position and velocity to produce good solutions to TSP in the PSO_TSP algorithm. This re-definition is needed to reflect the terrain that will be searched. The standard definitions work great in an integer or real-valued terrain, but have no meaning in the permutation terrain of the TSP. In PSO_TSP, positions represent potential solutions (a list of cities). A Velocity (V) is a list of permutations applied to a particle. Each particle has both a position and an associated velocity. The Velocity Best (VB) is a velocity that maps a particle to the best particle in the swarm found so far. In each generation, each particle is transformed to another particle using a combination of its V and VB. A “No-hope” condition is meant to detect early convergence to a local optimal solution. Possible “No-hope” conditions are all particles in the swarm within a given distance of the best solution, or a certain number of generations without discovering a new best solution. When a “No-hope” condition is met, the swarm is re-initialized to random positions to force more exploration. Thus, the Swarm moves and explores the search space according to Equations 1 and 2.

Equation 1 – Calculating a Single Particle’s New Velocity

$$V_{t+1} = C_1 V_t \oplus C_2 (P_{ig,t} - X_t) \oplus C_3 (P_{vg,t} - X_t)$$

Equation 2 – “Moving” a Single Particle in a Swarm

$$X_{t+1} = X_t + V_{t+1}$$

Where:

- 1) V_{t+1} – The particle’s new velocity for the next generation
- 2) C_1 – A percentage of the number of “steps” in a velocity to be used. A measure of how much the particle “trusts” its own exploration.
- 3) V_t – The particle’s current velocity. A list of permutation steps.
- 4) \oplus – The concatenation of velocity steps.
- 5) C_2 – A percentage of the number of “steps” in a velocity to be used. A measure of how much a particle “trusts” its neighborhood best velocity.
- 6) $P_{ig,t}$ – The neighborhood (from i to g) best position
- 7) “-” – The difference of two positions is the velocity that will transform the second position into the first position
- 8) X_t – The current position
- 9) C_3 – A percentage of the number of “steps” in a velocity to be used. A measure of how much a particle “trusts” the global velocity.
- 10) $P_{vg,t}$ – The global best position
- 11) “+” – The transformation of a position using the velocity (yields a position)
- 12) X_{t+1} – The particle’s new “moved” position. The position of the next generation.

2.2.1 PSO-TSP Algorithm

Following is an outline of the PSO_TSP algorithm:

1. Initialize the positions and velocities.
2. Determine new particles by applying V_{t+1} to each position (Equation 1,2).

3. Determine the best particle as the one with the shortest tour.
4. If the best solution in a neighborhood is a better solution than VB, save this solution as the new VB.
5. Perform steps 2-4 as above until there is no improvement in the global VB for a set number of iterations (other ending criterion may apply). At this point, we have converged to a local optima and have “No-hope” of finding better solutions. Several methods may be used to find better solutions such as simply starting again from step one without initializing the global best, or using a local search heuristic on the particles and performing steps 2-4.
6. Once all “No-hope” methods are exhausted, return the best solution found.

2.2.2 PSO_TSP Example Moving a Particle

Let’s look at an example of the way that PSO_TSP moves a single particle:

Let:

$$X_t = \{1,3,5,4,2,1\}$$

$$V_t = \{(3,5),(2,4)\}$$

$$C_1 = C_2 = C_3 = 0.5$$

$$P_{i,t} = \{1,3,4,5,2,1\}$$

$$P_{g,t} = \{3,4,2,1,5,3\}$$

From Equation 1:

$$\begin{aligned} V_{t+1} &= C_1 V_t + C_2 (P_{i,t} - X_t) + C_3 (P_{g,t} - X_t) \\ &= 0.5 \{(3,5),(2,4)\} + 0.5 \{(4,5)\} + \\ &0.5 \{(3,1),(4,1),(2,5)\} = \{(3,5)\} + \{(4,5)\} \\ &+ \{(3,1),(4,1)\} = \{(3,5),(4,5),(3,1),(4,1)\} \end{aligned}$$

Note that the list of permutations swaps cities, thus (3,5) means that city 3 is swapped with city 5.

From Equation 2:

$$X_{t+1} = X_t + V_{t+1} = \{3,1,4,5,2,3\}$$

2.3 Ant System (AS)

[Dorigo, Stutzle] describe an algorithm inspired by ants that can find solutions to TSP. In this algorithm, a population of ants creates a tour in a step-by-step process. When an ant is in a given city, it examines the pheromone levels leaving its current city and going to cities that the ant has not yet visited. The pheromone levels are used to determine the probability of the ant traveling to its next city. A higher pheromone level has a higher probability that the ant chooses that path. After all ants have completed a tour, the ant with the best fitness value is allowed to lay down pheromone. Variants typically involve the selection of ants that get to lay pheromone, the types of pheromone (reward and punishment), and the amount or weighting of pheromone used. Ant Colony Optimization (ACO) is an example of a variant based on pheromone weighting. In ACO, the trail with the highest pheromone level has a large weighting and is highly likely to be picked.

3 *PSO – Inspired by Ants* (PSO_AS)

3.1 Description

This algorithm combines the ideas presented previously. As in AS, the particles are moved in a step-by-step process to create a tour. Unlike AS, a table of pheromone levels is not kept. Instead, choices are based on the global best particle and the local best particle as in PSO_TSP. In fact, the algorithm is identical to that presented for PSO_TSP

(see 2.2.1), but the meaning behind applying the equations in step 2 alters the way the algorithm performs this step. When a particle is in a given city, it generates a random percentage. This percentage determines if the particle attempts to follow the global best, the local best, or takes a pseudo-random path based on the particle's position before commencing a new tour. If the global best is chosen, the cities in the global best (adjacent to the city the particle is currently in) become the first choice. In the case of asymmetric problems only one choice is allowed. If the global best choices are available (i.e., they have not yet been visited in this tour), one is chosen (at random) and the particle proceeds to that city. If there is no available global choice, or if local best is selected, the local best position is used to determine possible choices in the same way as the global best was as outlined. If the local best choices are not available, or if the "default" method is chosen, the most recently visited city (last if possible) from the particle's current position that has not yet been visited is chosen.

3.2 Redefinition of Equation 1 and 2 Terms

Equation 1 and Equation 2 define PSO and require the use of the global and local best in the search process. While PSO_AS uses Equation 1 and Equation 2, the following redefinition of terms is required to describe the new algorithm.

- 1) V_{t+1} – The particle's new velocity for the next generation. A summation of step-by-step choices that generates a new position.

- 2) C_1 - A probability of selecting to use information of the prior tour in creating the new tour. A measure of how much the particle "trusts" its own exploration.
- 3) V_t - The particle's current velocity. A particle's velocity and position contain redundant information.
- 4) \oplus - denotes local choices made based on the probabilities. $C_1 + C_2 + C_3 = 100\%$.
- 5) C_2 - A probability of selecting to use information of the neighborhood best. A measure of how much a particle "trusts" its neighborhood best.
- 11) "+" - The transformation of a position using the velocity (yields a position). Since the velocity contains redundant information with the position, this step is inherent.
- 12) X_{t+1} - The particle's new "moved" position. The position of the next generation.

3.3 Example: Moving a Particle

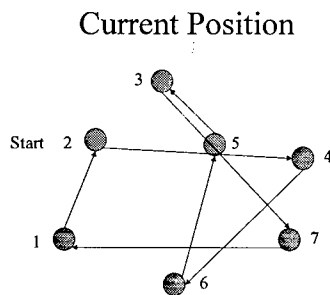


Figure 1 - X_t

- 6) $P_{ig,t}$ - The neighborhood (from i to g) best position
- 7) "-" - The difference of two positions is the velocity that will transform the second position into the first position. This velocity is equivalent to the first position, thus $P_{ig,t} - X_t = V_{ig,t}$.
- 8) X_t - The current position
- 9) C_3 - A probability of selecting to use information of the global best. A measure of how much a particle "trusts" its global best.
- 10) $P_{vg,t}$ - The global best position

Let's look at an example to illustrate how PSO_AS works.

Given a TSP problem where:

$$X_t = \{2, 4, 6, 5, 3, 7, 1\}$$

$$P_{vg,t} = \{3, 7, 6, 4, 5, 1, 2\}$$

$$P_{ig,t} = \{7, 6, 2, 3, 5, 4, 1\}$$

$$C_1 = 10\%$$

$$C_2 = 10\%$$

$$C_3 = 80\%$$

$$\text{Random numbers generated} = \{60, 82, 87, 26, 94\}$$

$$X_{t+1} = \{1, 2, 3, 5, 4, 7, 6\}$$

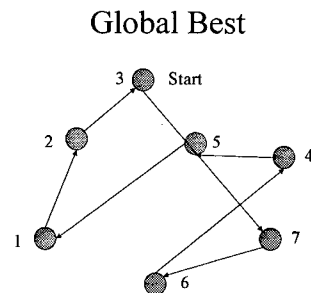


Figure 2 - $P_{vg,t}$

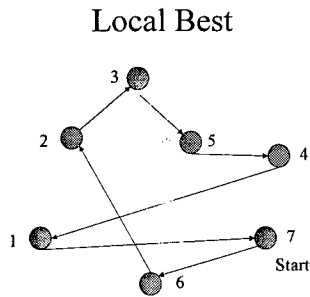


Figure 3 - $P_{ig,t}$

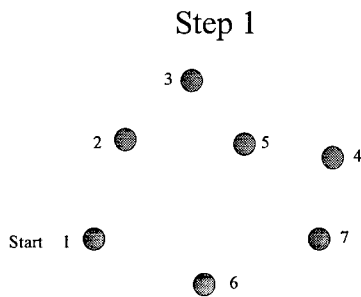


Figure 4 - Step 1

Step 1 (Figure 4): The default method is always used to select the starting city, thus city 1 is chosen since it is the last city in X_t .

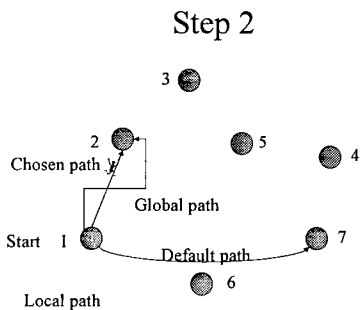


Figure 5 - Step 2

Step 2 (Figure 5): Since 60 is less than C_3 , the global information is used. City

2 is chosen since it follows city 1 in the global position. If the problem were symmetrical, city 5 may have been chosen instead since it precedes city 1.

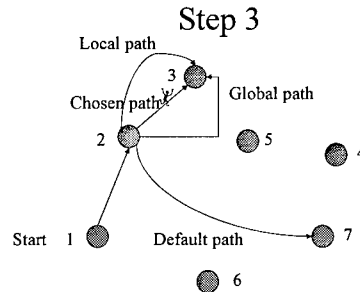


Figure 6 - Step 3

Step 3 (Figure 6): Since 82 is greater than C_3 but less than $C_3 + C_2$, the neighborhood information is used. City 3 is chosen since it follows city 2 in the neighborhood position. Note that when both the global and neighborhood positions contain the same paths, the probability of continuing to follow that path is equal to $C_3 + C_2$.

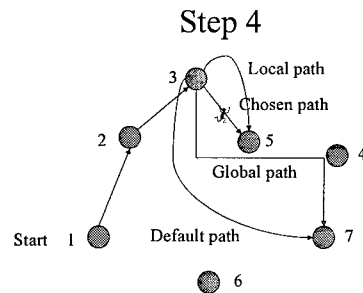


Figure 7 - Step 4

Step 4 (Figure 7): Since 87 is greater than C_3 but less than $C_3 + C_2$, the neighborhood information is used. City 5 is chosen since it follows city 3 in the neighborhood position.

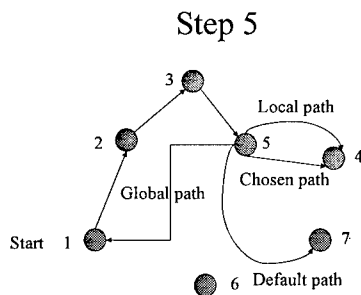


Figure 8 – Step 5

Step 5 (Figure 8): Since 26 is less than C_3 , the global information is used. City 1 follows city 5 in the global, but cannot be chosen since it has already been included in the tour (Step 1). We therefore attempt to use the neighborhood information. City 4 is chosen since it follows city 5 in the neighborhood position. If a neighborhood position's information cannot be used because it has already been included in the tour, no attempt is made to use the global information. Rather, the default information is used. Note that when neighborhood information is used, the probability of continuing the path when the global information has already been included in the tour is equal to $C_3 + C_2$, thus greatly increasing the chances of using the neighborhood information.

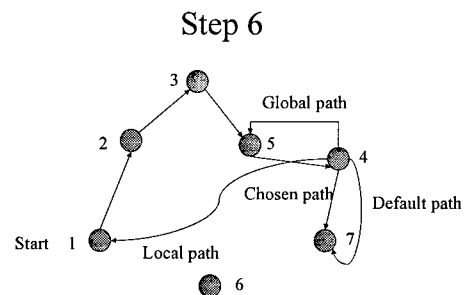


Figure 9 – Step 6

Step 6 (Figure 9): Since 94 is greater $C_3 + C_2$, the prior tour information or default is used. City 7 is chosen since it is the last in the list that has not already been included in the tour. City 1 was included in Step 1.

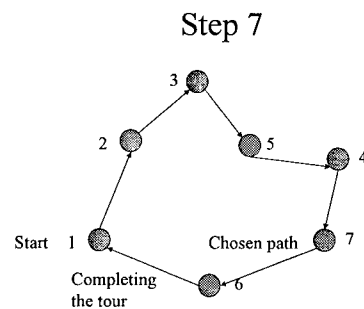


Figure 10 – Step 7; X_{t+1}

Step 7 (Figure 10): Since city 6 is the only city which has not been included in the tour, it is selected.

While the example demonstrates finding the solution, such is not necessarily the case. The new tour is built from particles that has survived, so the new tour is also likely to be good.

4 Experimental Approach

4.1 Purpose of Experiment

Since we have discussed two PSO algorithms, it may seem that we should compare and contrast them to determine which is the superior. That is not our purpose. We wish merely to point out that communication makes a difference in outcome, and to determine the properties of a Swarm to enable easier development of Swarm algorithms. Specific tuning of re-hope methods and re-hope/ convergence criterion can drastically alter results. Both quality and speed of solution are improved. We desired to observe both algorithms without these tuning parameters to assess each algorithm's ability to converge. Furthermore, we found that the implementation of PSO_TSP cannot handle problem sizes greater than 50 cities. This is due to the underlying structure causing stack overflow errors, not the algorithm. We therefore decided a competition would be pointless since problems of such small dimensionality are solved relatively easily by brute-force or heuristically by other methods such as Lin-Kernighan.

4.2 Design of Experiment

In order to meet the purposes of the experiment, both algorithms are executed with 16 particles and a neighborhood size of 4. These values are selected because they produced the best results in PSO_TSP [Clerc]. Except as noted below, all other parameters were default parameters recommended by PSO_TSP. Both randomly initialize the Swarm (PSO_TSP was slightly modified for this purpose). Both use the

global best and neighborhood best in moving the Swarm. Both stop after moving the swarm 600 times without any improvement in the global best solution. Neither use a re-hope method. PSO_AS used values of $C_1 = C_2 = 10\%$, and $C_3 = 80\%$. These values were chosen to exploit information of the global (C_3) and provide a fair amount of exploration (C_1). No work was performed to tune PSO_AS, so other values of C_1 , C_2 , and C_3 may produce better results. Three problems were selected at random from [TSPLIB], and one (br17.atsp) was selected because it was used by [Clerc]. All of these were executed five times to produce representative results. The metrics collected to demonstrate differences are current best solution, the time of finding the current best solution, and the number of iterations completed at the time that a new best solution is discovered.

5 Results & Analysis

5.1 Results

Table 1: Comparison of best Solutions.

File	PSO -TSP	PSO _AS	Optima
Br17.atsp	88	82	39
Bays29.tsp	2848	2067	2020
Ftv33.atsp	2261	1563	1286
P43.atsp	6056	5729	5620

Table 1 shows that PSO_AS converges to a better solution (see analysis). It should be noted that in all cases the worst solution was within 5% of the best solution. As mentioned previously, parameter tuning and re-hope methods improve the quality of solutions in both algorithms (PSO_AS typically finds the optima for these small problems in a

matter of milliseconds), but were purposefully excluded to compare the swarm's native ability to converge.

Table 2: Comparison of average time (sec) to converge and average iterations at convergence.

File	PSO-TSP		PSO_AS	
	Time	Iter.	Time	Iter.
Br17.atsp	22.4	372	.019	45
Bays29.tsp	21.8	327	0.19	178
Ftv33.atsp	37.8	451	0.56	428
P43.atsp	81.3	742	1.04	544

Table 2 shows that PSO_AS converges faster (fewer iterations) and is more timely than PSO_TSP for these TSP problems.

5.2 Analysis

Empirically, PSO_AS runs about 80 times faster per iteration than PSO_TSP.

PSO_AS also finds better solutions. Our early work with PSO_TSP (with parameter tuning and re-hope methods) produced good solutions, but not good enough (typically between 5 and 10% of the optima). To improve solutions, we examined how the swarm moved. Our conclusion was that PSO_TSP didn't exploit the information present in the global best solution or the local best solution well enough. Suppose we are moving a particle and fully trust the global best, don't trust the local best, and don't trust the individual position ($C_1 = 0$, $C_2 = 0$, $C_3 = 1$). In this case, moving the particle would result in the global best. Now, let's suppose we only partly trust the global solution ($C_1 = 0$, $C_2 = 0$, $C_3 = 0.8$). In this case, moving the particle would result in the first 80% of the cities being identical to those of the global best. Thus, moving a particle is

equivalent to a mutation operation on the global best particle where the mutation always occurs at the end of the position. If we were to let ($C_1 = .2$, $C_2 = 0$, $C_3 = .8$), we are then mutating even farther away. While PSO_TSP does exploit the information provided by the global best, there is no mechanism to allow it to exploit the right information.

This insight lead to the development of PSO_AS. It is specifically designed to exploit the information in the global and local best regardless of where that information may be. By randomly choosing to follow the global best at each node, it is likely that different particles will explore around the global in different areas.

What makes a swarm a swarm? If we look to nature such as bees or ants, we see many individuals that collectively perform "astounding" feats. Each individual contributes and performs their function without being aware of the overall goal. Certainly, each individual is not capable of producing solutions. Yet, what enables the swarm to produce its solution? Also inherent in a swarm (but less easily observed) is some form of communication between members that guides the individual in its choices. Bees can see each other as they fly [Insect]. The speed, direction, and position of nearby bees affect the flight path of individuals. Ants lay down pheromone to communicate past paths [Beckers]. Subsequent ants sense the pheromone levels and choose their path based on this local information.

Both PSO_TSP and PSO_AS are swarms. PSO requires particles in the swarm to "communicate" with the global and a local best particle (although other

swarms may have broader communication schemes). The difference between PSO_TSP and PSO_AS is not the communication pattern, but how that communication is being used. PSO_AS was designed to better exploit the information in the global and local best particle. With high probability, a good sequence in either the global or local particle remains intact.

6 Conclusions

A swarm requires some form of communication between members to affect the future actions of its members. The effectiveness of a swarm relies heavily on the relevance of the communication, and how the communication is exploited. When designing a swarm, pay particular attention to what information is communicated and how that information can be used and exploited to find good solutions. With PSO_AS and PSO_TSP, the information communicated is the global and local best solution found so far. PSO_AS exploits this information by building tours using many links from the global and local best solution.

It is also useful to look to nature for multiple methods of inspiration. Perhaps bees, ants, or even a flock of Geese has a communication method that will work best for your problem domain. PSO_TSP uses the communication similar to how bees do [Insect], with each particle altering its course toward the global and local best. While this approach may be superior for some problem domains (for example, target tracking [Carlisle]), it may not be the

best way to exploit the communication for a specific problem.

7 Future Work

We need to tune PSO_AS with many parameters (initialization, re-hope methods, ending criterion). We also will compare PSO_AS with heuristic and other stochastic methods available to demonstrate that PSO_AS is a viable, competitive TSP algorithm that has performance advantages.

References

- Beckers R., Deneuborg J.L. and Goss S., Trails and U-turns in the Selection of a Path of the Ant *Lasius Niger*, In *J. theor. Biol.* Vol. 159, pp. 397-415, 1992.
- Carlisle A., Dozier, G. Adapting Particle Swarm Optimization to Dynamic Environments. International Conference on Artificial Intelligence. 2000.
- Chapman, Richard and Moore. Application of Particle Swarm to Multiobjective Optimization. IEEE International Conference on Evolutionary Computation. 1999
- Clerc, M. (1999). The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization. Congress on Evolutionary Computation, Washington D.C., IEEE
- Clerc, Maurice. Discrete Particle Swarm Optimization Illustrated by the Travelling Salesman Problem. www.mauriceclerc.net. 2000. (Submitted for publication)

Dorigo M., V. Maniezzo and A. Colorni, The Ant System: An Autocatalytic Optimizing Process. Technical Report No. 91-016, Politecnico di Milano, Italy, 1991.

Dorigo M., T. Stutzle ACO Algorithms for the Traveling Salesman Problem. Evolutionary Algorithms in Engineering and Computer Science. John Wiley & Sons, 1999.

Insect Vision, Navigation, and "Cognition" Laboratory
<http://cvs.anu.edu.au/insect/insect.html>

Kenedy, J. and R.C. Eberhart (1995). Particle Swarm Optimization. IEEE International Conference on Neural Networks, Perth, Australia, IEEE Service Center, Piscataway, NJ.

Kennedy, J. (2000). Stereotyping: Improving particle swarm performance with cluster analysis. *Proceedings of the 2000 Congress on Evolutionary Computation*, 1507-1512.

Kennedy, J. (1999). Small worlds and mega-minds: Effects of neighborhood

topology on particle swarm performance. *Proceedings of the 1999 Conference on Evolutionary Computation*, 1931-1938.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinooy Kan and D. B. Shmoys, *The Travelling Salesman Problem*, Wiley, Chichester, 1985.

Tao, G. and Michalewicz, Z. Inver-over Operator for the TSP, Proceedings of the 5th Parallel Problem Solving from Nature, T. Baeck, A.E. Eiben, M. Schoenauer, and H.-P. Schwefel (Editors), Amsterdam, September 27-30, 1998, Springer-Verlag, Lecture Notes in Computer Science, pp.803-812.

TSPLIB <http://www.iwr.uni-heidelberg.de/iwr/comopt/software/TSPLIB95/>

White T., and Pagurek B., Towards Multi-Swarm Problem Solving in Networks, Proceedings of the 3rd International Conference on Multi-Agent Systems (ICMAS '98), July 1998.